Studies on Abductive and Nonmonotonic Reasoning(Dissertation_全文)

AUTHOR(S):

Inoue, Katsumi

CITATION:

Inoue, Katsumi. Studies on Abductive and Nonmonotonic Reasoning. 京都大学, 1993, 博士(工学)

ISSUE DATE:

1993-01-23

URL:

https://doi.org/10.11501/3064791

STUDIES ON ABDUCTIVE AND NONMONOTONIC REASONING

by

KATSUMI INOUE

October, 1992

STUDIES ON ABDUCTIVE AND NONMONOTONIC REASONING

by

KATSUMI INOUE

A dissertation submitted to
the Faculty of Engineering of
Kyoto University
for the degree of Doctor of Engineering

October, 1992

Preface

The major advantage of the logic approach to Artificial Intelligence is the ability to represent knowledge declaratively with logical formulas, thereby obtaining clear declarative semantics. However, there are two important difficulties in using classical logic to formalize commonsense knowledge. First, classical logic is monotonic, while commonsense reasoning is nonmonotonic. Second, classical logic is deductive, but commonsense reasoning often involves abductive inference.

In this dissertation, methodologies for abductive and nonmonotonic reasoning are studied within the logic approach. In particular, attention is paid to seeking answers to two important questions: "What are the relationships between abduction and nonmonotonic logics?", and "How can they be computed?"

In answer to the question on relationships, we show that the links between abduction and nonmonotonic reasoning are bidirectional; abduction can be formalized by using some nonmonotonic logics, and nonmonotonic logics can be computed by using abduction. This result also contributes to answering the second question.

As part of the second question, we investigate two ways of computing abduction and nonmonotonic reasoning.

On one hand, we characterize abduction in terms of a consequence-finding problem within first-order logic. This characterization enables efficient theorem proving techniques to be utilized for computation of abductive explanations. The proposed procedure, SOL-resolution, is both effective and complete for finding explanations. We also characterize theorem proving in circumscription as finding some combinations of abductive explanations, and develop an efficient mechanism for obtaining answers to queries.

On the other hand, we propose a framework of hypothetical reasoning within extended logic programs, for default reasoning and exceptions, inconsistency resolution, the closed world assumption, and abduction. We also show that such a framework can be transformed into an extended logic program, and that it can be computed by using a model generation theorem prover, based on fixpoint characterizations of such extended classes of logic programs.

Acknowledgments

I would like to express my sincere gratitude to my thesis advisor, Toshihide Ibaraki, for his continuous guidance, valuable suggestions and sharp criticism. I would also like to thank my other reading committee members of Kyoto University, Toshiharu Hasegawa and Makoto Nagao, for their valuable suggestions and encouragement.

This research was done as part of the R & D activities of the Fifth Generation Computer Systems (FGCS) project of Japan, and was conducted at the Institute for New Generation Computer Technology (ICOT). I wish to express my deep indebtedness to Kazuhiro Fuchi for providing me with a stimulating place to work.

Matsushita Electric Industrial Co., Ltd. supported my research in various ways. I am particularly grateful to Masato Yamazaki for providing me with an opportunity to work at ICOT and for his encouragement.

I am deeply indebted to my managers at ICOT; Koichi Furukawa and Ryuzo Hasegawa, for their helpful suggestions; and Yasuo Iwashita, Yuichi Fujii and Kenji Ikoma, for their encouragement. Their support was indispensable to my research.

I would like to thank my research partners in the Hypothetical Reasoning Group at ICOT, Yoshihiko Ohta and Makoto Nakashima, for their cooperation and discussions. My special thanks go to Jun Arima and Ken Satoh for their fruitful discussions and for teaching me logic. I have had many discussions about AI in general with Yasuo Nagai. Miyuki Koshimura provided me with the MGTP. Many other ICOT members have helped me in various ways, though I cannot list all their names.

Discussions with researchers outside ICOT and Matsushita are, also, always stimulating and helpful. Nicolas Helft and David Poole coauthored papers on theorem proving for circumscription with me. Various researchers have provided valuable comments on the work presented in this dissertation. In particular, I would like to thank Robert Demolombe, Michael Gelfond, Randy Goebel, Mitsuru Ishizuka, Kouji Iwanuma, Alex Kean, Bob Kowalski, Vladimir Lifschitz, Don Loveland, Ray Reiter, Chiaki Sakama and Mark Stickel.

Contents

Chapter 1

Introduction

> *An hypothesis is the holding for true of the judgment of the truth of a reason on account of the sufficiency of its consequents.*
>
> — Immanuel Kant.

In real world problem solving, we are often forced to draw conclusions even when complete information is not available. Modeling sophisticated agents capable of *reasoning with incomplete information* has been a major theme in the field of *Artificial Intelligence* (AI). This kind of reasoning is not only an advanced mechanism for intelligent agents to cope with some particular situations but an intrinsically necessary condition to deal with *commonsense reasoning*: it has been agreed that neither human beings nor computers can have all the information relevant to mundane or everyday situations. John McCarthy, a pioneer of the *logic approach* to AI, pointed out various important issues that arise in programming a computer to perform commonsense reasoning. For example, the *frame problem* [McCarthy and Hayes, 1969] stems from the need to represent information what remains unchanged under certain state changes. More generally, the *qualification problem* [McCarthy, 1977] arises in representing general commonsense knowledge: a general rule would have to include an impractical number of conditions if it should be interpreted as an accurate statement about the world. Yet, we humans can perform plausible inference that derives some conclusions despite the absence of such total knowledge about the world.

Besides commonsense reasoning, almost all intelligent systems should cope with incompleteness of information even in situations where knowledge should intuitively be complete. A typical example can be seen in database systems, which adopt the convention that facts that are not listed in a database are implicitly assumed to be false in the world. This convention, called the *closed world assumption* [Reiter, 1978], allows us to design database systems without including an impractically vast number of negative facts in databases.

To function without complete information, intelligent agents need to draw some *unsound* conclusions, or *augment* theorems, by applying such reasoning as *default reasoning* and *closed-world reasoning*. Conclusions in accordance with default reasoning are tentative: they are often retracted when new and contradictory information is acquired. Thus, this kind of reasoning is *nonmonotonic*: it does not hold that the more information we have the more consequences we get. This inference also has to anticipate the possibility of later revisions of beliefs.

A method of reasoning, called *hypothetical reasoning* [Inoue, 1988c], is a candidate for reasoning with incomplete information. It creates hypotheses, or assumptions, that some information would hold despite the absence of complete evidence for it, and proceeds to infer certain conclusions that are based on those hypotheses. Namely, the set of conclusions can be expanded by incorporating assumptions as long as there is no information to the contrary. This form of reasoning is actually used by human beings, and reflects the process of deriving conclusions based on a pattern of inference of the form: "Let's assume such and such and see what happens." Hypothetical reasoning is also a kind of nonmonotonic reasoning, because utilized hypotheses may be discarded later if they cause contradiction with a new fact.

Hypothetical reasoning is often used to give a certain *explanation* for some observed information. Such situation can be seen in seeking an answer to a question of the form: "Why does it happen?" or "Why did she say such a thing?" We can make assumptions about causes so that the observation can be actually derived as a conclusion of the hypotheses together with knowledge about the world. This inference to explanation is called *abduction* [Peirce, 1932], and has been a fundamental concept for various AI problems that deal with commonsense knowledge.

Having established the method of hypothetical reasoning, we can expect that various reasoning problems of interest in AI can be realized as computer programs. Thus, in this dissertation, we will concern ourselves with how to realize reasoning with incomplete information in terms of hypothetical reasoning. The principal keywords for this goal are *abduction* and *nonmonotonicity*.

1.1 Background

Logic can exert much influence on formalizing various kinds of reasoning. In particular, the logic approach to AI has the benefit that knowledge represented by logical formulas has a rigid mathematical foundation, and has clear declarative semantics. However, there are many difficulties in using classical logic to formalize commonsense or hypothetical reasoning. One of the most important drawbacks of classical logic is that it is *deductive* and is, thus, *monotonic*. On the other hand, as stated above,

hypothetical reasoning is *abductive* and *nonmonotonic*. Here, we touch on past studies on abduction and nonmonotonic reasoning. We will see a more detailed survey of them in Chapter 2.

Although we consider non-classical logic in this dissertation, some background in classical logic will be necessary for understanding the contents that follow. If the reader is not familiar with the predicate calculus, we recommend reading [Enderton, 1972], a textbook on first-order and second-order logics, or [Genesereth and Nilsson, 1987], an introduction to the logic approach of AI. Some familiarity with traditional theorem proving techniques for first-order logic would also be helpful for better understanding (see [Chang and Lee, 1973; Loveland, 1978] for an introduction).

1.1.1 Abductive Reasoning

In traditional logic, *modus ponens* is an inference rule [1] of the form:

$$\frac{A \quad A \supset C}{C}.$$

This rule means that, for any two first-order formulas, A and C, if whenever a set of first-order formulas contains A and $A \supset C$, then C results from applying modus ponens to the set. This pattern of reasoning is an instance of *deduction*, and is always sound and monotonic. On the other hand, consider the following unsound inference, called the *fallacy of affirming the consequent*, of the form:

$$\frac{C \quad A \supset C}{A}.$$

Charles Sanders Peirce [1932] coined the term *abduction* for this type of non-deductive inference. The point is that, although we cannot conclude that A is true, we can say that A is a sufficient *hypothesis* that *explains* C. The notion of explanation, or *abductive reasoning*, has been a fundamental concept for various AI problems. For example, in model-based diagnosis and synthesis such as plan recognition and design, when we observe the behavior of a system, we want to identify the hypotheses that, if combined with background knowledge of the system, can explain the observation [Pople, 1973; Cox and Pietrzykowski, 1986; Finger, 1987; Reiter, 1987]. Also, in natural language understanding, sophisticated user interfaces, and communication among intelligent agents, it is recognized that an explanatory capability may play a crucial role [Charniak and McDermott, 1985; Hobbs *et al.*, 1988].

[1] An *inference rule* has application conditions (the upper part of the rule), which, when satisfied, yield the formula which is the lower part of the rule. Application conditions of modus ponens consist of two schemata, each of which is replaced by a first-order formula.

In [Poole *et al.*, 1987], it was argued that hypothetical reasoning can be viewed as a process of *theory formation*; a set of possible hypotheses is accepted as part of a theory if the set can explain all the observations and is consistent with the given axioms. This is a *deductive-nomological* [Hempel, 1966] view of abduction, of which Peirce's theory of abduction was the forerunner [Inoue, 1992a]. Along this line, much literature has been produced on abductive reasoning [Reiter and de Kleer, 1987; Stickel, 1989; Levesque, 1989; Kakas and Mancarella, 1990; Console *et al.*, 1991].

Since hypotheses and formulas derived from them are not guaranteed to be true, if a contradiction occurs in the subsequent reasoning process, we must remove the original hypotheses and select others instead. Such an approach to consistency maintenance and belief revision in hypothetical reasoning, which has been used in practice, is found in the field of *truth maintenance systems* (TMS) [Martins, 1991]. A TMS is a subsystem attached to the main body of reasoning systems. It records statements about the inferences (*justifications*) performed so far by the reasoning system, and maintains the *consistency* of reasoned statements so that they can single out contradictions. Two typical types of TMSs have been proposed in the literature: a *justification-based TMS* (JTMS, or, simply, TMS) [Doyle, 1979] and an *assumption-based TMS* (ATMS) [de Kleer, 1986]. A JTMS labels each proposition with its status, which is either believed or disbelieved, and preserves the consistency of the belief set by keeping the status of the contradiction as always disbelieved. An ATMS labels each proposition with the sets of hypotheses that are needed to derive it, and preserves the consistency of the belief set by removing those sets of hypotheses that derive the contradiction from the labels of all propositions. Although truth maintenance and abductive reasoning have been investigated independently, these two have been unified in such as [Inoue, 1988a], and turn out to have common features.

1.1.2 Logic of Nonmonotonic Reasoning

As stated before, hypothetical reasoning is nonmonotonic. There have been many proposals since the beginning of the 80s on how to build nonmonotonic reasoning systems from some underlying logic. Research on such *nonmonotonic logics* can be broadly classified into the following two:

Adding a set of formulas to the premises.

This approach tries to build nonmonotonic reasoning within the predicate calculus, instead of extending the logic. Some formulas are added to the given set of premises so that the formulas are derived monotonically from the augmented theory. The nonmonotonicity arises from the fact that the formulas added to the premises may not grow monotonically as the premises grow. Examples are *closed-world assumptions* [Reiter, 1978; Minker, 1982; Gelfond *et al.*,

1989], *predicate completion* [Clark, 1978], and *circumscription* [McCarthy, 1980; McCarthy, 1986; Lifschitz, 1985].

Extending logical languages.

The notions of beliefs, consistency and/or default implication can be expressed by introducing modal operators, inference rules, and/or new connectives. The nonmonotonicity arises from the fact that those formulas with a kind of modality such as consistency and unprovability grow nonmonotonically. Typical examples can be seen in *default logic* [Reiter, 1980], *Nonmonotonic logic I* [McDermott and Doyle, 1980], *autoepistemic logic* [Moore, 1985], and recent theories of *logic programming* [Gelfond and Lifschitz, 1988]. These logics are sometimes called *nonmonotonic modal logics*.

1.2 Objectives

Hypothetical reasoning thus gets strong support from many branches of technical literature. However, we recognize that there are at least two fundamental issues that must be resolved for hypothetical reasoning in order to realize commonsense reasoning. These two issues form the following research directions.

First, the semantics of logics of hypothetical reasoning should be clearly provided to understand their properties. Although research activities on abductive reasoning and nonmonotonic logics have been pursued independently, the relationship between the two approaches should be clarified. Such relationships would enable us to discuss many different aspects of hypothetical reasoning in uniform frameworks.

Second, it is essential to answer the question of how abductive and nonmonotonic reasoning can be implemented. We would like to have not only their proof theories but also very efficient proof procedures for them. Note that, while mathematical bases of nonmonotonic logics have been investigated very well, some major logics, such as circumscription and nonmonotonic modal logics, have lacked good feasible procedural semantics.

Therefore, the main objective of this research is to formalize various kinds of hypothetical reasoning within simple and general frameworks, in the hope of enabling the applicable efficient computation techniques.

1.3 Contributions

The research reported in this dissertation contributes the following:

1. We show that the links between abduction and nonmonotonic reasoning are *bidirectional*; hypothetical reasoning can be formalized by using some nonmonotonic logics, and nonmonotonic logics can be computed by using hypothetical reasoning systems.

2. We introduce the notion of *characteristic clauses*, which is a useful theoretical framework for many reasoning procedures that should cope with incomplete information, and provide an effective mechanism called *SOL-resolution* for implementing the framework. As a consequence, we succeed in supplying an abductive procedure that is both complete and efficient.

3. We make a detailed analysis of proof procedures for *circumscription*. This analysis is also based on the notion of characteristic clauses.

4. We provide a theoretical foundation of hypothetical reasoning within the framework of *logic programming*, and give a simple procedure for its computation.

As a side effect of these contributions, we discover the usefulness of two interesting computational frameworks, *consequence-finding* and *model generation*, for solving various non-deductive reasoning problems. We also give such a unifying view of computational properties in Chapter 8.

1.4 Outline of the Dissertation

The dissertation consists of four parts. These parts in the organization basically correspond to the above four contributions.

1.4.1 Logical Framework for Hypothetical Reasoning

In Chapter 2, a detailed survey of existing hypothetical reasoning is given. It gives not only an overview but also some original results and observations that will be further explained in detail in subsequent chapters. We start with Peirce's logic of abduction, which is then related to a formalization of *explanation* within first-order logic. To know what formulas hold in the theory augmented by hypotheses, the notion of *prediction* is introduced. There are two approaches to nonmonotonic prediction: *credulous* (or *brave*) and *skeptical* (or *cautious*) approaches, depending on how conflicting hypotheses are treated. We also summarize the results on this aspect. It will be shown that abduction can be related to the brave approach, in particular to the simplest but most important subclass of Reiter's *default logic* for which efficient theorem proving techniques may exist. On the other hand, *circumscription* is a notable

example of the skeptical approach. Interestingly, we will see, in Chapter 5, that the skeptical approach can be implemented by using the brave approach.

1.4.2 Theorem Proving Approaches for Abduction

In Chapter 3, we discuss the evaluation properties of abduction in the context of the first-order predicate calculus. To make abduction and nonmonotonic reasoning (through abduction) computable, we try to re-evaluate the *consequence-finding* problem, which is an important problem in automated deduction [Lee, 1967; Slagle *et al.*, 1969; Minicozzi and Reiter, 1972]. The problem of consequence-finding is modified so that only interesting clauses with a certain property (called *characteristic clauses*) are found. Then, abduction is formalized in terms of characteristic clauses. This formalization also enables the characterization of many reasoning methods of interest to AI. In particular, the ATMS and its generalization (the *clause management system*, CMS) [Reiter and de Kleer, 1987] are precisely formalized by using the notion of characteristic clauses.

In Chapter 4, we consider a consequence-finding procedure for procedural semantics of abduction, by extending existing theorem proving methods. A familiar and efficient class of theorem proving techniques can be seen in *linear resolution with clause ordering* (C-ordered linear resolution) [Loveland, 1978; Kowalski and Kuhner, 1971]. However, C-ordered linear resolution has been used mainly for proof-finding, since it is incomplete for consequence-finding. We show that C-ordered linear resolution becomes complete for finding characteristic clauses if a *skip* rule is added to it. The proposed method is called *SOL-resolution*. Note that a formal proof of completeness for consequence-finding is quite hard to derive. Even if a strategy has been proved to be complete for refutation, it is not easy to prove that it is complete for consequence-finding. For instance, the thesis by Lee [1967] is entirely devoted to proving that the resolution principle [Robinson, 1965] is complete for consequence-finding, and the proof is far from obvious.

Compared with other resolution methods such as set-of-support resolution, SOL-resolution generates fewer clauses in order to find the characteristic clauses. SOL-resolution and its completeness result subsume a large part of the results that have already been presented in the literature of various fields. In the first-order case, SOL-resolution can be used for computing abduction. In the propositional case, this is an elegant tool for computing the *prime implicants/implicates* [Quine, 1952]. As an obvious application of the consequence-finding algorithm, we will also consider procedural semantics for a variety of ATMSs. Based on the definitions of the CMS/ATMS given in Chapter 3, we give algorithms for computing minimal explanations and for updating them.

1.4.3 Computing Circumscription by Using Abduction

Chapter 5 applies the consequence-finding methods defined in Chapters 3 and 4 to query-answering algorithms for first-order logic databases augmented with the *circumscription* axiom. Two recent proposals [Przymusinski, 1989; Ginsberg, 1989] for such algorithms are examined, and their merits are compared. We expand on the theoretical results of such algorithms from the viewpoint of abduction. Based on a detailed analysis of their computational properties, a more general procedure is proposed to save computational effort. Additionally, some limitations of these circumscriptive theorem provers are discussed.

We, then, address the problem of extracting answers in resolution-based theorem provers for circumscription. Analogously to answer-extraction techniques in first-order logic, the process extracts answer information from a proof of the query. However, circumscriptive theorem provers consist of two processes, one is to generate explanations for the theorem to be proven and the other is to show that such explanations cannot be refuted. In general, many explanations can be derived to support the theorem. We show that queries can be answered by finding certain combinations of explanations, present a procedure that searches the space of explanations, and show that it allows significant pruning in this space. These results are relevant to other nonmonotonic formalisms with explanation-based proof procedures.

1.4.4 Hypothetical Reasoning in Logic Programs

Chapter 6 returns to the subject of relating nonmonotonic logics with hypothetical reasoning. In particular, we address the relation with *logic programming* in the context of nonmonotonic modal logics. Extended logic programs proposed by Gelfond and Lifschitz [1990] are logic programs with *classical negation* along with *negation as failure*, and are capable of expressing incomplete knowledge. Their work is expanded in this chapter to deal with broader classes of commonsense knowledge. Like Theorist [Poole *et al.*, 1987], some rules are regarded as hypotheses, which are distinct from the theory about the world, and are used to augment the theory. This theory formation framework can be used for default reasoning, abduction, and inconsistency resolution. We also translate the framework to an extended logic program whose answer sets correspond to consistent belief sets of augmented theories. Note that a *nonmonotonic ATMS* [Dressler, 1989], which is an ATMS that accepts nonmonotonic justifications, can also be viewed as an abductive generalization of logic programs.

The results obtained in Chapter 6 indicate the importance of a tool that computes the *answer sets* of those extended logic programs. Chapter 7 gives a novel and simple procedure that computes the answer sets of *every* class of function-free logic programs and deductive databases containing negation-as-failure formulas. Such classes include

the *extended disjunctive databases* proposed by Gelfond and Lifschitz [1991] whose proof procedure has not been known so far. The proposal is based on the bottom-up, incremental, backtrack-free computation of the minimal models of positive disjunctive programs, together with integrity constraints over beliefs and disbeliefs. The translation method on which the procedure is based not only provides a simple fixpoint characterization of answer sets, but is also very helpful for understanding under what conditions each model is "stable" or "unstable". The procedure has been implemented using the model generation theorem prover MGTP [Fujita and Hasegawa, 1991] on a parallel inference machine, and has been applied to a legal reasoning system.

In Chapter 8, the dissertation concludes with a summary of the achievements in this field and the discussion of some important future work.

1.5 Publications

The publications on which each chapter of this dissertation is based are as follows.

Chapter 2 is partially based on the survey paper published in the *Journal of Japanese Society for Artificial Intelligence* in January 1992 [Inoue, 1992a].

Chapters 3 and 4 are mainly based on the paper published in *Artificial Intelligence*, Vol. 56, Nos. 2–3 [Inoue, 1992b], which is an extended version of the paper presented at the *Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91; Sydney, Australia, August 1991) [Inoue, 1991b]. Also, some parts of these chapters present the results included in the paper published in *Proceedings of the 1990 Workshop on Truth Maintenance Systems* (Stockholm, Sweden, August 1990), Lecture Notes in Artificial Intelligence Vol. 515, Springer-Verlag [Inoue, 1990b].

Chapter 5 is mainly from two joint papers, one with Nicolas Helft which was presented at the *Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (CSCSI-90; Ottawa, Ontario, Canada, May 1990) [Inoue and Helft, 1990], and the other with Nicolas Helft and David Poole which was presented at the *Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91; Sydney, Australia, August 1991) [Helft *et al.*, 1991].

Chapter 6 is mainly based on an extended version of the paper presented at the *Eighth International Conference on Logic Programming* (ICLP '91; Paris, France, June 1991) [Inoue, 1991a].

Chapter 7 is based on a joint paper with Miyuki Koshimura and Ryuzo Hasegawa which was presented at the *Eleventh International Conference on Automated Deduction* (CADE-11; Saratoga Springs, NY, June 1992) [Inoue *et al.*, 1992a].

It should be also noted that the dissertation does not include all of the author's work on hypothetical reasoning. Hypothetical reasoning is one of the reasoning mechanisms that span a wide range of theory and practice. Throughout the dissertation,

theoretical aspects of abduction and nonmonotonic proof procedures are focused on. However, equally important are applications of such technologies to practice through efficient implementations of hypothetical reasoning systems. For implementation issues of such practical hypothetical reasoning systems, the reader is referred to the author's papers on controlling problem solvers with hypotheses [Inoue, 1988b; Inoue, 1988c; Inoue and Ohta, 1990], and the joint papers with Yoshihiko Ohta on forward-chaining hypothetical reasoning systems with the ATMS (called APRICOT) [Ohta and Inoue, 1990; Ohta and Inoue, 1991a; Ohta and Inoue, 1991b; Ohta and Inoue, 1992]. By the experience of such a pragmatic approach to hypothetical reasoning, the author now believes that hypothetical reasoning is a fundamental technology in new generation computing, whose applications range from commonsense reasoning to scientific and engineering programming.

Chapter 2

Nonmonotonic Reasoning and Abduction

In this chapter, we review existing frameworks for abduction and nonmonotonic reasoning. We concentrate on those logical frameworks whose computational properties will be investigated in subsequent chapters. We shall begin with the logical framework of abduction, then move on to some existing nonmonotonic logics. Finally, we shall explain the relationships between abduction and nonmonotonic reasoning.

2.1 Logic of Abduction

Abduction has recently been recognized as a very important form of reasoning in AI. It is one of the three fundamental modes of reasoning characterized by Peirce [1932], the others being *deduction* and *induction*. To see the differences between these three reasoning modes, let us look at the "beans" example used by Peirce [1932, paragraph 623] in a syllogism form. Abduction amounts to concluding the minor premise (*Case*) from the major premise (*Rule*) and the conclusion (*Result*):

(*Rule*)	All the beans from this bag are white.
(*Result*)	These beans are white.
(*Case*)	These beans are from this bag.

On the contrary, deduction amounts to concluding *Result* from *Rule* and *Case*, and induction amounts to concluding *Rule* from *Case* and *Result*. The form of abductive inference is written symbolically as follows.

> The (surprising) fact, C, is observed;
> But if A were true, C would be a matter of course;
> Hence, there is reason to suspect that A is true.

Therefore, the rule of the form:

$$\frac{C \quad A \supset C}{A}, \tag{2.1}$$

introduced in Chapter 1, can be obtained. Both abduction and induction are non-deductive and generate hypotheses. However, hypothesis generation by abduction is clearly distinguished from that by induction, in the sense that while induction infers something to be true through generalization of a number of cases of which the same thing is true, abduction can infer something quite different from what is observed. Therefore, according to Peirce [1932, paragraph 777], abduction is "the only kind of reasoning which supplies new ideas, the only kind which is, in this sense, synthetic." Since abduction can be regarded as a method to explain observations, Peirce considered it as the basic method for *scientific discovery*.

In the above sense, abduction is "amplicative" reasoning and may play a key role in the process of advanced inference. For example, analogical reasoning can be formalized by abduction plus deduction [Peirce, 1932, paragraph 513] (see also [Arima, 1992]). It is, however, only "probable" inference as it is non-deductive. That is, as Peirce argues, abduction is "a weak kind of inference, because we cannot say that we believe in the truth of the explanation, but only that it may be true". This phenomenon of abduction is preferable, since our *commonsense reasoning* also has a probable nature. In everyday life, we regularly form hypotheses, to explain how other people behave or to understand a situation, by filling in the gaps between what we know and what we observe. Thus, abduction is a very important form of reasoning, in everyday life as well as in science and engineering.

Another important issue involved in abduction is the problem of *hypothesis selection*: what is the best explanation, and how can we select it from a number of possible explanations which satisfy the rule (2.1)? Peirce considered this problem philosophically, and suggested various preference criteria that are both qualitative and economical. One example of such criteria is the traditional maxim of *Occam's razor*, which adopts the simplest hypotheses.

In the following subsections, we give a logic of abduction studied in AI from two points of views, namely, explanation and prediction.

2.1.1 Explanation

Firstly, we shall connect Peirce's logic of abduction with formalizations of abduction within first-order logic developed in AI. The most popular formalization of abduction in AI defines an explanation as a set of hypotheses which, together with the background theory, logically entails the given observations [Charniak and McDermott,

1985]. This *deductive-nomological* view of explanation [Hempel, 1966] has enabled logical specifications of abduction and their proof procedures based on the *resolution principle* [Robinson, 1965]. There are a number of proposals for resolution-based abductive systems [Pople, 1973; Cox and Pietrzykowski, 1986; Finger, 1987; Poole *et al.*, 1987; Reiter, 1987; Reiter and de Kleer, 1987; Stickel, 1989; Inoue, 1990b; Inoue, 1991b; Demolombe and Fariñas del Cerro, 1991; Console *et al.*, 1991]. [1]

In the deductive-nomological view of explanation, the following assumptions are made:

1. Knowledge about a domain of discourse, or *background knowledge*, can be represented by a set of first-order formulas as the *proper axioms*. In the following, we denote such an axiom set as Σ, and call it a set of *facts*.

2. For an observation C expressed by a first-order formula, each explanation A of C satisfying the rule (2.1) can be constructed from a sub-vocabulary $\mathcal{H}$ of the representation language that contains Σ. We call each formula constructed from such a subset of the language a *hypothesis*. In general, a hypothesis constructed from $\mathcal{H}$ is a formula whose truth value is indefinite but may be assumed to be true. Sometimes $\mathcal{H}$ is the representation language itself.

3. The major premise $A \supset C$ in the rule (2.1) can be obtained *deductively* from Σ, either as an axiom contained in Σ or as a *logical consequence* of Σ:

$$\Sigma \models A \supset C\,. \tag{2.2}$$

4. Σ contains all the information required to judge the acceptability of each hypothesis A as an explanation of C. That is, each formula A satisfying (2.2) can be tested for its appropriateness without using information not contained in Σ. One of these domain-independent, necessary conditions is that A should not be contradictory to Σ, or that $\Sigma \cup \{A\}$ is *consistent*.

5. We adopt Occam's razor as a domain-independent criterion for hypothesis selection. Namely, the *simplest* explanation is preferred over any other.

[1] Other formalizations of abduction can be found in AI literature. For example, abduction is defined by the *set covering model* in [Bylander *et al.*, 1991], and is discussed at the *knowledge level* in [Levesque, 1989]. For a summary of abductive frameworks, see [Inoue, 1992a] or [Kakas *et al.*, 1992]. Levesque's [1989] formulation also suggests that abduction does not have to be formalized within first-order logic. There are some proposals for abductive frameworks based on other logical languages such as logic programming [Kakas and Mancarella, 1990; Inoue, 1991a] and modal logics [Siegel and Schwind, 1991] (see Section 2.3.3).

These postulates are useful particularly for domain-independent automated abduction. The first and second conditions above define a logical framework of abduction: the facts and the hypotheses are both first-order formulas. The third and fourth conditions give a logical specification of the link between the observations and explanations: theories augmented with explanations should both entail observations and be consistent. Although these necessary conditions are most common in abductive frameworks proposed in AI, the correctness of them from the philosophical viewpoint is still being argued. The fifth condition, simplicity, is also one of the most agreeable criterion to select explanations: a simpler explanation is preferred if every other condition is equal in multiple explanations. Note that these conditions are only for the definition of explanations. Criteria for *good*, *better*, or *best* explanations are usually given using meta information and domain-dependent heuristics. A number of factors should be considered in selecting the most reasonable explanation. Since there has been no concrete consensus among AI researchers or philosophers about the preference criteria, we will not discuss them further in the following chapters.

As an example of a logical framework for abductive reasoning which satisfies all the above postulates, we introduce *Theorist* proposed by Poole, Goebel and Aleliunas [1987], which is a simple hypothetical reasoning system based on a first-order theorem prover that distinguishes facts from hypotheses. Additional Theorist background is given by Goebel, Furukawa and Poole [1986], Poole [1988; 1989; 1991], Lin and Goebel [1989], and Satter, Goodwin and Goebel [1990].

Definition 2.1 (Theorist) Let Σ be a set of facts, and Γ a set of hypotheses. Given a closed formula G, a set E of ground instances of elements of Γ is an *explanation* of G with respect to (Σ, Γ) [2] if

1. $\Sigma \cup E \models G$, and
2. $\Sigma \cup E$ is consistent.

An explanation G is *minimal* if no proper subset E' of E is an explanation of G.

The first condition in the above definition reflects the fact that Theorist has been devised for automated scientific *theory formation*, which is intended to be very useful for prototyping many AI problem solving systems by providing a simple "hypothesize-test" framework, i.e., *hypothetical reasoning*. However, since we can identify a set of formulas with the conjunction of them, when an explanation is $E = \{H_1, \ldots, H_n\}$, it is easy to see that the first condition is equivalent to

$$\Sigma \models H_1 \wedge \ldots \wedge H_n \supset G\,,$$

[2] Some Theorist literatures such as [Poole, 1988] give a slightly different version of the definition, where a set $\Sigma \cup E$ (*scenario*) satisfying the two conditions is called an explanation of G.

and thus can be written in the form of (2.2). The minimality criterion is a syntactical form of Occam's razor. Since for an explanation E of G, any $E' \subseteq E$ is consistent with Σ, the condition can be written as: E is a minimal explanation of G if no $E' \subset E$ satisfies $\Sigma \cup E' \models G$. Note that in Theorist, explanations are defined as a set of ground instances. We will give a more general definition of (minimal) explanations in Section 3.3.1 in which variables can be contained in explanations.

Example 2.2 Suppose that the facts and the hypotheses are given as:

$$\begin{aligned}\Sigma_1 = \{ \quad & \forall x\,(\,Bird(x) \wedge \neg Ab(x) \supset Flies(x)\,)\,, \\ & \forall x\,(\,Penguin(x) \supset Ab(x)\,)\,, \\ & Bird(Tweety) \quad \}\,, \\ \Gamma_1 = \{ \quad & \neg Ab(x) \quad \}\,.\end{aligned}$$

Here, the hypothesis $\neg Ab(x)$ means that for any ground term t, $\neg Ab(t)$ can be hypothesized. In other words, a hypothesis containing variables is shorthand for the set of its ground instances. Intuitively, $\neg Ab(x)$ means that anything can be assumed to be not abnormal (i.e., normal). In this case,

$$\text{a minimal explanation of } Flies(Tweety) \text{ is } \{\, \neg Ab(Tweety)\,\}\,,$$

because $\neg Ab(Tweety)$ is a ground instance of $\neg Ab(x)$, and

1. $\Sigma_1 \cup \{\neg Ab(Tweety)\} \models Flies(Tweety)$, and
2. $\Sigma_1 \cup \{\neg Ab(Tweety)\}$ is consistent.

In Theorist, a pre-specified set Γ of hypotheses can be any set of first-order formulas. However, Poole [1988] shows a *naming* method which transforms each hypothesis in Γ into an atomic formula. The naming method converts the facts-hypotheses pair (Σ, Γ) into an equivalent but simpler pair (Σ', Γ') in the following way. For every hypothesis $F(\mathbf{x})$ in Γ, where $\mathbf{x} = x_1, \ldots, x_n$ is the tuple of the free variables [3] appearing in F, we associate a predicate symbol δ_F not appearing anywhere in (Σ, Γ), and define the following sets of formulas:

$$\begin{aligned}\Gamma' &= \{\, \delta_F(\mathbf{x}) \mid F(\mathbf{x}) \in \Gamma\,\}\,, \\ \Sigma' &= \Sigma \cup \{\, \forall \mathbf{x}\,(\,\delta_F(\mathbf{x}) \supset F(\mathbf{x})\,) \mid F(\mathbf{x}) \in \Gamma\,\}\,.\end{aligned}$$

Then, there is a 1-1 correspondence between the explanations of G with respect to (Σ, Γ) and the explanations of G with respect to (Σ', Γ') [Poole, 1988, Theorem 5.1].

[3] An occurrence of a variable in a formula is *bound* if the occurrence either immediately follows a quantifier symbol or is within the scope of a quantifier of the same name. A variable is *free* in a formula if at least one of its occurrences is not bound in the formula.

Example 2.2 (continued) The hypothesis $\neg Ab(x)$ can be named $Normal(x)$:

$$\Sigma_1' = \Sigma \cup \{\ \forall x\,(\,Normal(x) \supset \neg Ab(x)\,)\ \}\,,$$
$$\Gamma_1' = \{\ Normal(x)\ \}\,.$$

In this case, a minimal explanation of $Flies(Tweety)$ is $\{\,Normal(Tweety)\,\}$, which corresponds to the explanation $\{\,\neg Ab(Tweety)\,\}$ with respect to the original (Σ_1, Γ_1).

Naming hypotheses is a technique commonly used in most abductive systems because hypotheses in the form of atomic formulas can be processed very easily in their implementation. In particular, we can utilize well-known efficient linear procedures (see Sections 2.3.1 and 4.2.1). Restriction of hypotheses to atoms is thus used in many systems such as [Finger, 1987; Stickel, 1989; Kakas and Mancarella, 1990; Console *et al.*, 1991]. Furthermore, in Chapter 6, we will use a similar naming technique for a theory formation framework based on extended logic programs. Note that when we use a resolution procedures for non-Horn clauses, we can allow for *negative* as well as positive *literals* as names of hypotheses, since both positive and negative literals can be resolved upon in the procedure. In fact, we will define an abductive framework in Section 3.3.1 in a way that gives the hypotheses as a set of literals. Other examples of abductive systems that allow literal hypotheses can be seen in such as [Pople, 1973; Cox and Pietrzykowski, 1986]. When the hypotheses are given as a set of literals, we can deal with them just as they are. For Example 2.2, we do not have to rename the negative literal $\neg Ab(x)$ to the positive literal $Normal(x)$ (see Section 2.3.1 for how this example can be computed). This kind of negative abnormal literal was originally used by McCarthy [1986] (see, also, Example 2.6), and thus is very convenient for computing circumscription through abduction (see Chapter 5).

2.1.2 Prediction

Theory formation frameworks like Theorist can be used for *prediction* as well as abduction. In [Poole, 1989], a distinction between explanation and prediction is discussed as follows. Let Σ be a set of facts, Γ a set of hypotheses, G a formula, and E an explanation of G with respect to (Σ, Γ) as defined by Definition 2.1.

1. In abduction, G is an observation which is known to be true. We may assume E is true because G is true.

2. In prediction, G is a formula or a query whose truth value is unknown but is expected to be true. We may assume E is true to make G hold under E.

Both of the above ways of theory formation perform hypothetical reasoning, but in different ways. In abduction, hypotheses used to explain observations are called *conjectures*, whereas, in prediction, hypotheses are called *defaults* [Poole, 1988; Poole, 1989]. In Example 2.2, if we have observed that $Tweety$ was flying and we want to know why this observation could have occurred, then obtaining the explanation $E_1 = \neg Ab(Tweety)$ is abduction; but if all we know is only the facts Σ_1 and we want to know whether $Tweety$ can fly or not, then finding E_1 is prediction where we can expect $Tweety$ may fly by default reasoning. These two processes may occur successively: when an observation is made, we abduce possible hypotheses; from these hypotheses, we predict what else we can expect to be true. In such a case, hypotheses can be used as both conjectures and defaults.

A hypothesis regarded as a default may be used unless there is evidence to the contrary. Therefore, defaults may be applied as many as possible unless augmented theories are inconsistent. This leads to the notion of *extensions* [Poole, 1988].

Definition 2.3 Given the facts Σ and the hypotheses (defaults) Γ, an *extension* of (Σ, Γ) is the set of logical consequences of $\Sigma \cup \mathbf{M}$ where $\mathbf{M}$ is a maximal (with respect to set inclusion) set of ground instances of elements of Γ such that $\Sigma \cup \mathbf{M}$ is consistent.

By using the notion of extensions, various alternative definitions of what should be predicted can be given [Poole, 1989; Lin and Goebel, 1989]. They are related to the *multiple extension problem*: if G_1 holds in an extension X_1 and G_2 holds in another extension X_2, but there is no extension in which both G_1 and G_2 hold (i.e., $X_1 \cup X_2$ is inconsistent), then what should we predict? –Nothing? Or both (or one of) G_1 and G_2? The next two are the most well-known prediction policies:

1. Predict what holds in an extension of (Σ, Γ);

2. Predict what holds in all extensions of (Σ, Γ).

The first approach to default reasoning leads to multiple extensions and is called a *credulous* (or *brave*) approach. On the other hand, the latter approach is called a *skeptical* (or *cautious*) approach.

We will see later that credulous prediction can be directly characterized by explanation (Section 2.3.1) and that skeptical prediction can be represented by combining explanations (Section 2.3.2). Before that, we shall summarize more general nonmonotonic formalisms for both credulous and skeptical default reasoning.

2.2 Logical Frameworks for Nonmonotonic Reasoning

Several formalisms for nonmonotonic reasoning were studied in depth during the last decade and are now relatively well understood. This section reviews three major formalisms for nonmonotonic reasoning: *default logic* [Reiter, 1980], *circumscription* [McCarthy, 1980], and *negation as failure* in logic programming [Clark, 1978; Gelfond and Lifschitz, 1988]. Many other important formalizations are omitted in this section, including *closed world assumptions* [Reiter, 1978; Minker, 1982; Gelfond *et al.*, 1989], *predicate completion* [Clark, 1978], *Nonmonotonic logic I* [McDermott and Doyle, 1980], and *autoepistemic logic* [Moore, 1985; Konolige, 1988; Marek and Truszczyński, 1989]. The reader is referred to these publications for these other logics and their relationships.

Each of default logic and circumscription extends the classical first-order predicate calculus, but in a different way. Default logic introduces *inference rules* referring to the *consistency* with a belief set, and uses them meta-theoretically to extend a first-order theory. Circumscription, on the other hand, augments a first-order theory with a *second-order axiom* expressing a kind of *minimization* principle, and restricts the objects satisfying a certain predicate to just those that the original theory says must satisfy that predicate. Although these two major formalisms are based on quite different backgrounds, we will see in a later section (Section 2.3.2) that there are interesting relationships between them.

2.2.1 Default Logic

Default logic proposed by Reiter [1980] is a logic for drawing plausible conclusions based on consistency. This is one of the most intuitive and natural logics for nonmonotonic reasoning. One of the most successful results derived from the studies on default logic can be seen in its two important applications:

1. *Abduction* can be characterized by one of the simplest classes of default logic (see Section 2.3.1).

2. *Logic programs* can be interpreted as a class of default logic (see Section 2.2.3).

A *default* is an inference rule of the form:

$$\frac{\alpha(\mathbf{x}) \ : \ \mathsf{M}\,\beta_1(\mathbf{x}), \ldots, \mathsf{M}\,\beta_m(\mathbf{x})}{\gamma(\mathbf{x})}\,, \tag{2.3}$$

where $\alpha(\mathbf{x})$, $\beta_1(\mathbf{x}), \ldots, \beta_m(\mathbf{x})$, and $\gamma(\mathbf{x})$ are first-order formulas whose free variables are contained in a tuple of variables $\mathbf{x}$. $\alpha(\mathbf{x})$ is called the *prerequisite*, $\beta_1(\mathbf{x}), \ldots, \beta_m(\mathbf{x})$

the *justifications*, and $\gamma(\mathbf{x})$ the *consequent* of the default. A default is *closed* if no formula in it contains a free variable; otherwise it is *open*. An open default is usually identified with the set of closed defaults obtained by replacing the free variables with ground terms. The intended meaning of the default (2.3) is: for any tuple $\mathbf{t}$ of ground terms, "if $\alpha(\mathbf{t})$ is inferable and each of $\beta_1(\mathbf{t}), \ldots, \beta_m(\mathbf{t})$ is consistently assumed, then infer $\gamma(\mathbf{t})$".

A default is *normal* if it contains only one justification that is equivalent to the consequent; otherwise it is *nonnormal*. For example, the statement that "normally a bird can fly" can be expressed by the normal default:

$$\frac{Bird(x) \,:\, \mathsf{M}\, Flies(x)}{Flies(x)} \,.$$

A *default theory* is a pair, (D, W), where D is a set of defaults and W is a set of first-order formulas. The formulas of W represent a standard first-order proper axiom set. For example, the fact, $Bird(Tweety)$, and the exception to flight,

$$\forall\, x\, (\, Penguin(x) \supset \neg Flies(x)\,)\,,$$

are given as formulas in W.[4] A default theory is *normal* if every default is normal.

When a default is applied, it is necessary that each of its justifications is consistent with a "belief set". In the above example, since $Bird(Tweety)$ is a proper axiom, a belief set may contain $Flies(Tweety)$, the consequent of the default, together with the axioms W, and it is verified that it is consistent to assume $Flies(Tweety)$, the justification of the default, in the belief set. In order to express this condition formally, an acceptable "belief set" induced by reasoning with defaults should be defined precisely. In default logic, such a resultant belief set is called an *extension* of a default theory and is defined as follows:

Definition 2.4 [Reiter, 1980] Let (D, W) be a default theory, and X a set of formulas. X is an *extension* of (D, W) if it coincides with the smallest set Y of formulas satisfying the following three conditions:

1. $W \subseteq Y$.
2. Y is *deductively closed*, that is, it holds that $cl(Y) = Y$, where for any first-order theory T, $cl(T)$ is the logical closure of T under classical first-order deduction.
3. For any ground instance of any default in D of the form (2.3), if $\alpha(\mathbf{t}) \in Y$ and $\neg\beta_1(\mathbf{t}), \ldots, \neg\beta_m(\mathbf{t}) \notin X$, then $\gamma(\mathbf{t}) \in Y$.

[4] Each formula in W can be considered as a special default whose prerequisite is the formula *true* and whose justification is empty ($m = 0$ in the form (2.3)) [Gelfond *et al.*, 1991].

Reiter [1980] presented an alternative definition of extensions as follows.

Proposition 2.5 [Reiter, 1980, Theorem 2.1] A set of formulas X is an extension of a default theory (D, W) if and only if

$$X = \bigcup_{i=0}^{\infty} X_i \,,$$

where $X_0 = W$ and for any $i \geq 0$,

$$\begin{aligned} X_{i+1} = cl(X_i) \cup \\ \{\, \gamma(\mathbf{t}) \mid \alpha(\mathbf{t}) \in X_i \text{ and } \neg\beta_1(\mathbf{t}), \ldots, \neg\beta_m(\mathbf{t}) \notin X \\ \text{for a ground instance of a default in } D \text{ of the form (2.3)} \,\}\,. \end{aligned}$$

□

Notice that the extension X appears in both of its definitions in Definition 2.4 and Proposition 2.5. In other words, a default extension is defined by a *fixpoint equation*, and thus cannot be defined constructively. Moreover, there are default theories with multiple or, even, no extensions. However, it is known that for any *normal default theory*, there is at least one extension [Reiter, 1980, Theorem 3.1].

It is also noted that in default logic each extension is interpreted as an acceptable set of beliefs in accordance with default reasoning. Such an approach to default reasoning leads to multiple extensions and is a *credulous* approach. By credulous approaches one can get more conclusions depending on the choice of the extension so that conflicting beliefs can be supported by different extensions. This behavior is not necessarily intrinsic to a reasoner dealing with a default theory; we could define the theorems of a default theory to be the intersection of all its extensions so that we remain agnostic to conflicting information. This latter approach is a *skeptical* approach. Circumscription shown in the next subsection is an example of skeptical approaches to default reasoning.

2.2.2 Circumscription

Circumscription, proposed by McCarthy [1980], is one of the most "classical" and best-developed formalisms for nonmonotonic reasoning. It has been observed in the literature that an important property of circumscription that other commonsense representation formalisms lack, is that it is based on classical predicate logic. In this subsection, we give a brief analysis of circumscription. Additional background can be found in [McCarthy, 1977; McCarthy, 1980; McCarthy, 1986; Lifschitz, 1985].

Let T be a set of first-order formulas, and $\mathbf{P}$ and $\mathbf{Z}$ denote disjoint tuples of distinct predicate symbols in the language of T. The predicates in $\mathbf{P}$ are said to be *minimized* and those in $\mathbf{Z}$ to be *variables*; $\mathbf{Q}$ denotes the rest of the predicates in the language of T, called the *fixed* predicates (or *parameters*). We denote a theory T by $T(\mathbf{P};\mathbf{Z})$ when we want to indicate explicitly that T mentions the predicates $\mathbf{P}$ and $\mathbf{Z}$. Adopting the formulation by Lifschitz [1985], [5] the *circumscription of* $\mathbf{P}$ *in* T *with* $\mathbf{Z}$, written $CIRC(T;\mathbf{P};\mathbf{Z})$, is the augmentation of T with a second-order axiom expressing the minimality condition:

$$T(\mathbf{P};\mathbf{Z}) \wedge \neg\exists\, \mathbf{pz}\, (T(\mathbf{p};\mathbf{z}) \wedge \mathbf{p} < \mathbf{P})\,. \tag{2.4}$$

Here, $\mathbf{p}$ and $\mathbf{z}$ are tuples of predicate variables each of which has the same arity as the corresponding predicate symbol in $\mathbf{P}$ and $\mathbf{Z}$, and $T(\mathbf{p};\mathbf{z})$ denotes a theory obtained from $T(\mathbf{P};\mathbf{Z})$ by replacing each occurrence of $\mathbf{P}$ and $\mathbf{Z}$ with $\mathbf{p}$ and $\mathbf{z}$. Also, $\mathbf{p} < \mathbf{P}$ stands for the conjunction of formulas each of which is defined, for every member P_i of $\mathbf{P}$ with a tuple $\mathbf{x}$ of object variables and the corresponding predicate variable p_i in $\mathbf{p}$, in the form:

$$\forall \mathbf{x}\, (p_i(\mathbf{x}) \supset P_i(\mathbf{x})) \wedge \neg\forall \mathbf{x}\, (P_i(\mathbf{x}) \supset p_i(\mathbf{x}))\,.$$

Thus, the second-order formula in the definition (2.4) represents that the extension of the predicates from $\mathbf{P}$ is minimal in the sense that it is impossible to make it smaller without violating the condition T. Intuitively, $CIRC(T;\mathbf{P};\mathbf{Z})$ is intended to minimize the number of objects satisfying $\mathbf{P}$, even at the expense of allowing more or different objects to satisfy $\mathbf{Z}$.

Example 2.6 Let T consist of the three formulas:

$$\begin{aligned}
&\forall\, x\, (Bird(x) \wedge \neg Ab(x) \supset Flies(x))\,,\\
&\forall\, x\, (Penguin(x) \supset Bird(x))\,,\\
&\forall\, x\, (Penguin(x) \supset \neg Flies(x))\,.
\end{aligned}$$

Here, the predicate symbols in the theory are divided into: $\mathbf{P} = Ab$, $\mathbf{Z} = Flies$, and $\mathbf{Q} = Bird, Penguin$. The first formula is intended to express that normal (i.e., not Ab-normal) birds can fly. The abnormality with respect to flying is expressed by the predicate Ab, which is to be minimized in order to allow as few abnormal

[5] In Lifschitz's formulation, $T(\mathbf{P};\mathbf{Z})$ can be any second-order formula, and $\mathbf{Z}$ can be a tuple of distinct object, function and/or predicate constants. See [Enderton, 1972] for introduction of second-order logic. However, if $\mathbf{Z}$ is allowed to include object/function constants, then circumscription may not preserve satisfiability even for *universal* formulas (or, prenex form of universally quantified first-order formulas), that is, $CIRC(T;\mathbf{P};\mathbf{Z})$ may have no models even if a universal theory T is satisfiable [Lifschitz, 1986].

birds as possible. This "abnormality theory" is introduced by McCarthy [1986] for its uses in commonsense reasoning. In this example, the second-order axiom in the circumscription (2.4) can be written as

$$\forall\, ab', flies'\, \neg\, (\, T(ab'; flies') \wedge \forall x(ab'(x) \supset Ab(x)) \wedge \neg\forall x(Ab(x) \supset ab'(x))\,)$$

so that we have two universally quantified predicate variables ab' and $flies'$. Now, let us make the following substitutions for these variables:

$$ab'(x) \equiv Penguin(x)\,,$$
$$flies'(x) \equiv Bird(x) \wedge \neg Penguin(x)\,.$$

By these substitutions and the theory $T(Ab; Flies)$ in (2.4), we can derive the formula

$$\forall\, x\, (Ab(x) \equiv Penguin(x))$$

in first-order logic. Namely, the only abnormal birds with respect to ability to fly are penguins. From this it follows easily by a first-order deduction that

$$\forall\, x\, (Bird(x) \wedge \neg Penguin(x) \supset Flies(x))\,.$$

Notice that the last formula is not entailed by the original (uncircumscribed) theory T alone.

The model-theoretic characterization of circumscription is based on the following definition and result.

Definition 2.7 Let M_1 and M_2 be models of T with the same universe. We write $M_1 \leq_{\mathbf{P},\mathbf{Z}} M_2$ if M_1 and M_2 differ only in the way they interpret predicates from $\mathbf{P}$ and $\mathbf{Z}$, and the extension of (the relation corresponding to) every predicate P from $\mathbf{P}$ in M_1 is a subset of the extension of P in M_2.
A model M of T is $(\mathbf{P}, \mathbf{Z})$-*minimal*[6] if, for no other model M' of T, $M' \leq_{\mathbf{P},\mathbf{Z}} M$ but $M \not\leq_{\mathbf{P},\mathbf{Z}} M'$.

Theorem 2.8 [Lifschitz, 1985] For any formula F, $CIRC(T; \mathbf{P}; \mathbf{Z}) \models F$ if and only if F is satisfied by every $(\mathbf{P}, \mathbf{Z})$-minimal model M of T.

Since each theorem of a circumscription is satisfied by all minimal models, this property makes the behavior of circumscription skeptical.

[6] We will often say just *minimal models*, when $\mathbf{P}$ and $\mathbf{Z}$ are clear from the context.

2.2.3 Logic Programming

Historically, *logic programs* have been syntactically defined as sets of, usually Horn, clauses, and their semantics have been given by the *minimal model semantics*. Here, as in the case of circumscription, we use the notion of minimal models, but in this case the minimized predicates **P** are all predicates in the language and only Herbrand interpretations are compared (see [Bossu and Siegel, 1985]). If a logic program Σ is a set of Horn clauses, then its meaning is determined by the intersection of all models of Σ, which is the unique minimal model of Σ [van Emden and Kowalski, 1976]. Note that the unique minimal model (the least model) semantics is consistent with the *closed world assumption* [Reiter, 1978]: [7] a formula is not satisfied by the minimal model if and only if it is not a logical consequence of the program. Thus, as long as a logic program is defined as a set of first-order formulas, it can be regarded as a special case of circumscription (see also [Minker, 1982] for non-Horn programs).

However, once the notion of negation is explicitly introduced in the conditional part of a formula in logic programs, the situation changes because such a use of negation is extralogical. This kind of negation is called *negation as failure* [Clark, 1978] and is clearly distinguished from negation in classical first-order logic (*classical negation*). A formula in a program containing negation as failure is therefore called a *rule* instead of a clause. Much recent literature has been concerned with the declarative meaning of logic programs with negation as failure (see, for example, [Bidoit, 1989, Chapter 5] for a survey). Among the many proposed semantics, we choose here the *answer set semantics* proposed by Gelfond and Lifschitz [1988; 1990; 1991].

A *general logic program* is a set of rules of the form:

$$A \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n\,, \tag{2.5}$$

where $n \geq m \geq 0$, and each of A and A_i $(1 \leq i \leq n)$ is an atom. If a rule contains variables, it is *open*; otherwise it is *closed*. An open rule is usually considered to be the set of closed rules obtained by replacing the variables with ground terms.

The *answer sets* (or *stable models*) of a general logic program are defined in the following two steps [Gelfond and Lifschitz, 1988]:

1. Let Σ be a set of closed rules not containing *not*. The *answer set*, $\alpha(\Sigma)$, of Σ is the smallest set S of ground atoms satisfying the condition:

   For any rule $A \leftarrow A_1, \ldots, A_m$ from Σ, if $A_1, \ldots, A_m \in S$, then $A \in S$.

[7] The CWA defined by Reiter augments the theory by adding every negative literal $\neg P$ such that the ground atom P is not entailed by the theory.

2. Let Π be any set of closed rules. A set S of ground atoms is an *answer set* of Π if S is the answer set of Π^S, that is, $S = \alpha(\Pi^S)$, where Π^S is the set of rules without *not* obtained from Π by removing

   (a) every rule containing a formula *not* A in its body with $A \in S$, and

   (b) every formula *not* A in the bodies of the remaining rules.

According to Bidoit and Froidevaux [1989, Chapter 4] and Gelfond and Lifschitz [1990], an answer set of a general logic program Π can be interpreted as the atoms of an extension of a default theory $(\Pi, \emptyset)$ if we identify each rule of the form (2.5) in Π with a default

$$\frac{A_1 \wedge \ldots \wedge A_m \,:\, \mathsf{M}\,\neg A_{m+1}, \ldots, \mathsf{M}\,\neg A_n}{A}\,.$$

Therefore, an alternative characterization of answer sets can be given as follows. A set of ground atoms S is an answer set of a general logic program Π if and only if S coincides with the smallest set S' of ground atoms satisfying the condition:

> For any ground instance of any rule in Π of the form (2.5),
> if $A_1, \ldots, A_m \in S'$ and $A_{m+1}, \ldots, A_n \notin S$, then $A \in S'$.

Relationships between logic programs and other nonmonotonic formalisms have also been investigated. For example, Gelfond and Lifschitz [1988] and Elkan [1990] show a 1-1 correspondence between the answer sets and the expansions of the corresponding *autoepistemic theory* [Moore, 1985]. Also, a clear relationship with the *(justification-based) TMS* [Doyle, 1979] has been reported in [Elkan, 1990]. However, once *classical negation* is reintroduced to programs (called *extended logic programs*), along with negation as failure, to handle incomplete information, the relation with default logic turns out to be more appropriate and more natural than relations with other logics (see [Gelfond and Lifschitz, 1990]). Gelfond and Lifschitz [1991] further consider the class of *(extended) disjunctive databases* to allow for a form of disjunction, and such a database can be viewed as a *disjunctive default theory* [Gelfond *et al.*, 1991]. We will consider these extended classes of logic programs and their properties together with procedures which compute their answer sets in Chapters 6 and 7.

Possible-world semantics have also been proposed for negation as failure. For example, Lin and Shoham [Lin, 1991] use the logic GK built upon a *preferential nonmonotonic logic* [Shoham, 1987] so that logic programs can be formalized in *circumscription* with some constraints, and Lifschitz [1991] gives meaning to epistemic queries (i.e., *what a logic program knows*) in the context of logic programming.

2.3 Relating Abduction to Nonmonotonic Logics

In this section, we relate the abductive frameworks introduced in Section 2.1 to the formalisms of nonmonotonic reasoning shown in Section 2.2.

Since abduction is amplicative and plausible reasoning, conclusions of abductive reasoning may not be correct. Therefore, abduction is nonmonotonic. This can be easily verified for the Theorist framework; even if an explanation E is *consistent* with the facts Σ, E is not an explanation with respect to the new facts Σ' ($\supset \Sigma$) because $\Sigma' \cup E$ may not be consistent. Moreover, even if we do not require the consistency of explanations (note that this is different from Theorist), a *minimal* explanation E of G with respect to Σ may not be minimal with respect to Σ' ($\supset \Sigma$) because a subset E' of E may satisfy $\Sigma' \cup E' \models G$. Poole [1989] investigates other possibilities of nonmonotonicity that may arise according to changes of facts, hypotheses, and observations.

The above discussion can also be verified by considering relationships between abduction and nonmonotonic logics. We will see that the links are bidirectional: abduction can be formalized by a credulous form of nonmonotonic logic (*default logic*, Section 2.2.1), and a skeptical nonmonotonic formalism (*circumscription*, Section 2.2.2) can be represented using an abductive framework. The former relationship verifies the nonmonotonicity of abduction, and the latter implies that abduction can be used for commonsense reasoning as well as scientific theory formation.

2.3.1 Abduction and Normal Default Theory

Recall that a *normal default theory* [Reiter, 1980] is a pair, (D, W), where D is a set of normal defaults and W is a set of formulas. Suppose that Σ is a set of facts and Γ is a set of hypotheses. In this section, in order to avoid confusion in terminology, we call an extension of the Theorist framework (Σ, Γ) given by Definition 2.3 a *Theorist extension*, and call an extension of a default theory (D, W) given by Definition 2.4 a *default extension*. Let $w(\mathbf{x})$ be a formula whose free variables are $\mathbf{x}$. For Σ and Γ, we define a normal default theory (D_Γ, Σ), where

$$D_\Gamma = \left\{ \left. \frac{: \mathsf{M}\, w(\mathbf{x})}{w(\mathbf{x})} \right| w(\mathbf{x}) \in \Gamma \right\}.$$

Notice that D_Γ is a set of *prerequisite-free* normal defaults, that is, normal defaults whose prerequisites are *true*. The next two properties are given by Poole [1988].

Lemma 2.9 [Poole, 1988, Theorem 2.6] Let G be a formula. There is an explanation of G with respect to (Σ, Γ) if and only if there is a Theorist extension of (Σ, Γ) in which G holds. □

Lemma 2.10 [Poole, 1988, Theorem 4.1] A set X of formulas is a Theorist extension of (Σ, Γ) if and only if X is a default extension of (D_Γ, Σ). In this case, X is the smallest set of formulas satisfying the conditions: (a) $\Sigma \subseteq X$, (b) $cl(X) = X$, and (c) for any ground instance w of any element of Γ, if $\neg w \notin X$, then $w \in X$. □

Here, we obtain the next theorem by combining the above two lemmas.

Theorem 2.11 There is an explanation of G with respect to (Σ, Γ) if and only if G holds in a default extension of the default theory (D_Γ, Σ). □

Theorem 2.11 is very important with respect to the following two points:

1. It is verified that each abductive explanation is contained in a possible set of beliefs. In particular, when the hypotheses Γ represent defaults for normal or typical properties, then in order to predict a formula G by *default reasoning*, it is sufficient to find an explanation of G with respect to (Σ, Γ) [Poole, 1988].

2. All properties possessed by normal default theories are valid for abductive explanations and Theorist extensions. For instance, for any Σ and Γ, there is at least one Theorist extension of (Σ, Γ). Moreover, computation of abduction can be given by *top-down default proofs* [Reiter, 1980], which is an extension of linear resolution theorem proving procedures such as [Loveland, 1978; Reiter, 1971; Kowalski and Kuhner, 1971; Chang and Lee, 1973]. [8]

The second point above is also very useful for designing and implementing hypothetical reasoning systems. In fact, most first-order abductive procedures [Pople, 1973; Cox and Pietrzykowski, 1986; Poole *et al.*, 1987; Stickel, 1989; Poole, 1991] can be regarded as variants of Reiter's top-down default proof procedures: computation of explanations of G from (Σ, Γ) can be seen as an extension of proof-finding by introducing a set of hypotheses from Γ that, if they could be proven by preserving the consistency of the augmented theories, would complete the proofs of G. Alternatively, abduction can be characterized by a *consequence-finding* problem [Inoue, 1991b; Inoue, 1992b], in which some literals are allowed to be hypothesized (or *skipped*) instead of proven, so that new theorems consisting of only those skipped literals are derived at the end of deductions instead of just deriving the empty clause (see Chapter 4). In this sense, abduction can be implemented as an extension of deduction, in particular of a top-down, backward-chaining theorem-proving procedure. [9]

[8] This fact holds for the following reasons. It is shown that, G holds in some default extension of a normal default theory (D, W) if and only if there is a top-down default proof of G with respect to (D, W) [Reiter, 1980, Theorem 7.3]. Also, every top-down default proof *returns* a set S of instances of defaults of D with which G can be proven from W, i.e., $W \cup S \models G$. Therefore, such an S is an explanation with respect to the corresponding Theorist framework if $W \cup S$ is consistent.

[9] However, there is nothing to prevent us from using a bottom-up, forward-reasoning procedure to implement abduction. The abductive reasoning system APRICOT/0 [Ohta and Inoue, 1990;

Example 2.2 (continued) For the goal $G = Flies(Tweety)$, a version of Theorist implementation works as follows (written using Prolog-like notations):

$$\begin{array}{l} \leftarrow Flies(Tweety)\,, \\ \leftarrow Bird(Tweety) \wedge \neg Ab(Tweety)\,, \\ \leftarrow \neg Ab(Tweety)\,, \\ \Box \qquad \text{by defaults: } \{\neg Ab(Tweety)\}\,. \end{array}$$

Then, the returned set of defaults $S = \{\neg Ab(Tweety)\}$ is checked for the consistency with Σ_1 by failing to prove the negation of the conjunction of elements of S from Σ_1. In this case, it holds that

$$\Sigma_1 \not\models Ab(Tweety)\,,$$

thus showing that S is an explanation of G with respect to (Σ_1, Γ_1). Suppose that $Penguin(Tweety)$ is then added to Σ_1 and that

$$\Sigma_2 = \Sigma_1 \cup \{\ Penguin(Tweety)\ \}\,.$$

We then get S again by the same top-down default proof as above, but the consistency check of S in this case results in a different proof:

$$\begin{array}{l} \leftarrow Ab(Tweety)\,, \\ \leftarrow Penguin(Tweety)\,, \\ \Box\,. \end{array}$$

Therefore, S is no longer an explanation of G with respect to (Σ_2, Γ_1).

From the first remark for Theorem 2.11, a credulous approach to default prediction can be handled just by finding an explanation of a query with respect to the corresponding abductive framework. Note, however, that this property holds only for prerequisite-free normal default theories; it does not hold even for normal default theories (with prerequisites). Consider, for example, the default:

$$\frac{Bird(x)\ :\ \mathsf{M}\, Flies(x)}{Flies(x)}\,.$$

By creating the name $Normal(x)$, the following formula may be added to the facts:

$$\forall\, x\,(\,Bird(x) \wedge Normal(x) \supset Flies(x)\,)\,.$$

Ohta and Inoue, 1992] is such an example. See also Section 8.3.1.

The default hypothesis $Normal(x)$ thus corresponds to the prerequisite-free normal default:

$$\frac{: \mathsf{M}\, Bird(x) \supset Flies(x)}{Bird(x) \supset Flies(x)}$$

by Lemma 2.10. Now, suppose that the formula $\neg Flies(Sam)$ is in the facts. Then, the default $Normal(Sam)$ may derive $\neg Bird(Sam)$, but this formula is not contained in a default extension of the original default theory (by the asymmetry of default rules). In order to prevent Theorist from applying contrapositives of default conclusions, the following constraint is introduced in [Poole, 1988]:

$$\forall\, x\, (\, \neg Flies(x) \supset \neg Normal(x)\,)\, .$$

Note that the last formula can only be used as a *constraint* that should be satisfied by each Theorist extension, and cannot be dealt with as a fact because if it was everything could be predicted to fly without being proven to be a bird!

Poole [1988] also attempts an approximation of nonnormal default theories by using abduction. He translates a default of the form:

$$\frac{\alpha(\mathbf{x}) \,:\, \mathsf{M}\, \beta_1(\mathbf{x}), \ldots, \mathsf{M}\, \beta_m(\mathbf{x})}{\gamma(\mathbf{x})}\, ,$$

into the naming defaults $\delta_{\beta_i}(\mathbf{x})$ for every $i = 1, \ldots, m$ and the fact:

$$\forall\, \mathbf{x}\, (\, \delta_{\beta_1}(\mathbf{x}) \wedge \ldots \wedge \delta_{\beta_m}(\mathbf{x}) \wedge \alpha(\mathbf{x}) \supset \gamma(\mathbf{x})\,)\, .$$

The condition that, for any ground term $\mathbf{t}$, $\beta_i(\mathbf{t})$ has to be consistent with the theorems is then simulated by blocking the applicability of the hypothesis $\delta_{\beta_i}(\mathbf{t})$ with the *constraint*:

$$\forall\, \mathbf{x}\, (\, \neg \beta_i(\mathbf{x}) \supset \neg \delta_{\beta_i}(\mathbf{x})\,)\, .$$

Unfortunately, this translation is not exact for default theories in general. Although, in many cases, we could represent knowledge within a (modified) Theorist framework instead of using either nonnormal defaults or normal defaults with prerequisites, there are also many cases in which those general defaults are quite useful for nonmonotonic reasoning and representing commonsense knowledge. For example, the translation of logic programs with negation as failure into default theories (shown in Section 2.2.3) provides an elegant declarative semantics for logic programs. For another example, nonnormal defaults are used for eliminating unintended models in formal theories of reasoning about action that involve nonmonotonic persistence axioms (or, the law of inertia) [Morris, 1987]. As Morris [1987] argues, the core of the notorious *Yale shooting problem* [Hanks and McDermott, 1986] that, if formalized by normal defaults

or circumscription, supports two interpretations of the events where only one makes intuitive sense, can be reformulated using the following two propositional nonnormal defaults:

$$D_{YSP} = \left\{ \frac{: \mathsf{M} \neg Q}{P}, \quad \frac{: \mathsf{M} \neg R}{Q} \right\}.$$

For the default theory $(D_{YSP}, \emptyset)$, there is only one default extension, $cl(\{ Q \})$, which does not contain P. However, if we use Poole's method, the two defaults are transformed into the hypotheses $\{ \delta_{\neg Q}, \delta_{\neg R} \}$, the facts:

$$\delta_{\neg Q} \supset P, \quad \delta_{\neg R} \supset Q,$$

and the constraints:

$$Q \supset \neg\delta_{\neg Q}, \quad R \supset \neg\delta_{\neg R}.$$

We then obtain two maximal consistent sets of hypotheses satisfying the constraints, $\{\delta_{\neg R}\}$ and $\{\delta_{\neg Q}\}$, which correspond to the two Theorist extensions (one containing Q, and the other containing P). Therefore, as [Hanks and McDermott, 1986] shows, the problem cannot be expressed properly by a (prerequisite-free) normal default theory in an obvious way. Although [Morris, 1987] also uses a TMS [Doyle, 1979] formulation, the above problem can be expressed by the following logic program:

$$P \leftarrow not\, Q,$$
$$Q \leftarrow not\, R.$$

It is easy to see that $\{Q\}$ is the only answer set of the program. Therefore, nonnormal defaults are dealt with more naturally by negation as failure than by abduction. [10] In Section 2.3.3, we will see how negation as failure can be simulated by abduction. However, computation of such a simulation by first-order abduction is again only an approximation and is not exact. In fact, as Reiter [1980] argues, there is no local property in evaluating nonnormal default theories.

2.3.2 Abduction and Circumscription

At first glance, abduction and circumscription look very different. This relationship can be reduced to that of prerequisite-free normal default theories and circumscription since we have connected abduction and default logic in the last subsection. Thus, we

[10] If each prerequisite is a conjunction of literals, and all justifications and consequents are literals, formulation by an extended logic program is straightforward [Gelfond and Lifschitz, 1990]. If they are general first-order formulas, we may use the translation of defaults into a TMS network by [Junker and Konolige, 1990], which can be identified with an extended logic program.

shall consider here the translation from circumscription to such a default theory. Let us first consider some sufficient conditions for such a translation to work.

The most significant difference between circumscription and default logic lies in the ways to handle variables and equality. Consider, for example, the case where the predicate P is minimized in the set:

$$T = \{\ P(A)\ \}\ .$$

Then, the following formula can be obtained by the circumscription:

$$\forall\, x\, (\, P(x) \equiv x = A\,)\,, \tag{2.6}$$

that is, *every object* except A does not satisfy P. In contrast, to make $P(x)$ false as many as possible, consider the default theory (D, T) where

$$D = \left\{ \frac{:\ \mathsf{M}\,\neg P(x)}{\neg P(x)} \right\}\ .$$

The theorems predicted by this default theory are the literals $\neg P(t)$ for the set of *all ground terms* t different from A. This result is not equivalent to the fact that no objects different from A satisfy P. To fill in the gap between these results, we make the assumption that the only objects in the domain are the ones that can be named using the object and function constants in the language. This is called the *domain-closure assumption* (DCA) and can be written using the following domain-closure axiom [Reiter, 1984]:

$$\forall x\ (\ x = t_1 \vee x = t_2 \vee \dots\ )\,,$$

where the t_i's are all ground terms. Note that if there exist either an infinite number of object constants or function symbols other than constants, then the DCA is no longer first-order expressible because there are infinite terms. Now, suppose that the constant symbol B is contained in the language. Then, the default extension of (D, T) contains $\neg P(B)$, and thus $A \neq B$ is concluded. However, the circumscription of P in T plus the DCA again gives the formula (2.6) from which $\neg P(B)$ cannot follow unless $A \neq B$ is proven. Therefore, we need another assumption that ground terms can be assumed to be unequal if they cannot be proven equal. This is called the *unique-names assumption* (UNA). Under the DCA and the UNA, the circumscription of P in T yields the same result as the unique default extension of (D, T).

From the above discussion, in the following we assume that the function symbols are the constants only (it is often stated that the language does not contain function symbols) and the number of constants is finite. From now on, a theory T means a set of formulas over the language including the equality axioms, and both

the DCA and the UNA are assumed to be satisfied by T. In this case, the UNA represents that each pair of distinct constants denotes different individuals in the domain. The DCA implies that the theory has finite models and that every formula containing variables is equivalent to a propositional combination of ground atoms. While these assumptions are very strong, the importance of these assumptions is widely recognized in databases [Reiter, 1984] and logic programming [Lloyd, 1984; Lloyd and Topor, 1984]. For circumscription, these assumptions make the universe fixed so that the comparison with default logic becomes clear [Etherington, 1988]. In particular, circumscription with these assumptions is essentially equivalent to the *Extended Closed World Assumption* (ECWA) [Gelfond *et al.*, 1989].

Another big difference between circumscription and default logic is in their approaches to default prediction: skeptical versus credulous. The theorems of a circumscription are the formulas satisfied by every minimal model, while there are multiple default extensions in default logic. We, therefore, compare the theorems of a circumscription with the formulas contained in every default extension of a default theory.

On the relationship between circumscription and default logic, Etherington [1988] has shown that, under some conditions, a formula is entailed by a circumscription plus the DCA and the UNA if and only if the formula is contained in every default extension of the corresponding default theory.

Proposition 2.12 [Etherington, 1988] Assume that T is a theory satisfying the above conditions. Let $\mathbf{P}$ be a tuple of predicates, and $\mathbf{Z}$ the tuple of all predicates (other than those in $\mathbf{P}$) in the language. Then, those formulas true in every default extension of the default theory:

$$\left(\left\{ \frac{: \mathsf{M} \neg P_i(\mathbf{x})}{\neg P_i(\mathbf{x})} \;\middle|\; P_i \in \mathbf{P} \right\}, \; T \right) \tag{2.7}$$

are precisely those theorems of $CIRC(T; \mathbf{P}; \mathbf{Z})$. □

Since the default theory (2.7) is a prerequisite-free normal default theory, we can connect each of its default extensions with a Theorist extension using Lemma 2.9. Therefore, in the abductive framework, we can hypothesize the negative occurrences of the minimized predicates $\mathbf{P}$. The following corollary can be obtained by Theorems 2.8 and 2.11.

Corollary 2.13 Let T, $\mathbf{P}$ and $\mathbf{Z}$ be the same as in Proposition 2.12. A $(\mathbf{P}, \mathbf{Z})$-minimal model of T satisfies a formula F if and only if F has an explanation from the abductive framework $(T, \{ \neg P_i(\mathbf{x}) \mid P_i \in \mathbf{P} \})$.

The above corollary does not deal with a skeptical prediction but a credulous one. Moreover, Proposition 2.12 does not allow for the specification of *fixed predicates*. Gelfond *et al.* [1989], on the other hand, show a more general result for the ECWA by allowing some predicates to be fixed. The idea of reducing circumscription to the ECWA is very important as it is helpful for designing theorem provers for circumscription [Przymusinski, 1989; Ginsberg, 1989; Baker and Ginsberg, 1989; Inoue and Helft, 1990; Helft *et al.*, 1991]. Earlier work for such a reduction of circumscription to a special case of the ECWA can be found in [Minker, 1982; Bossu and Siegel, 1985] where all the predicates in the language are minimized.

To compute circumscription, we are particularly interested in the two results of the ECWA obtained by Gelfond *et al.* [1989, Theorems 5.2 and 5.3] with the notion of *free for negation*. These are also adopted as the basic characterizations for query answering in circumscriptive theories by Przymusinski [1989, Theorems 2.5 and 2.6]. Inoue and Helft [1990] express them using different terminology (*characteristic clauses*), which will be shown as Theorems 5.1 and 5.2 in Chapter 5. Here, we shall relate these results of the ECWA with abduction. Let T be a theory as above, $\mathbf{P}$ the minimized predicates, $\mathbf{Q}$ the fixed predicates and $\mathbf{Z}$ variables. For a tuple $\mathbf{R}$ of predicates in the language, we denote by $\mathbf{R}^+$ ($\mathbf{R}^-$) the positive (negative) occurrences of predicates from $\mathbf{R}$ in the language. Then, we define the abductive framework for circumscription, (T, Γ_{circ}), where the hypotheses are given as:

$$\Gamma_{circ} = \mathbf{P}^- \cup \mathbf{Q}^+ \cup \mathbf{Q}^- .$$

Intuitively, both positive and negative occurrences of $\mathbf{Q}$ can be hypothesized as defaults to prevent the abductive framework from altering the definition of each predicate from $\mathbf{Q}$. The next theorem is the result of a detailed analysis of [Gelfond *et al.*, 1989, Theorems 5.2 and 5.3] which we present in Chapter 5.

Theorem 2.14 [Inoue and Helft, 1990]
(1) For any formula F not containing predicate symbols from $\mathbf{Z}$,
$CIRC(T; \mathbf{P}; \mathbf{Z}) \models F$ if and only if $\neg F$ has no explanation from (T, Γ_{circ}). □
(2) For any formula F, $CIRC(T; \mathbf{P}; \mathbf{Z}) \models F$ if and only if there exist explanations $E_1, \ldots, E_n$ $(n \geq 1)$ of F from (T, Γ_{circ}) such that

$$\neg(E_1 \vee \ldots \vee E_n)$$

has no explanation from (T, Γ_{circ}). □

By using Theorem 2.14, we can reduce query answering in a circumscriptive theory to the finding of a combination of explanations of a query such that the negation of the disjunction cannot be explained. The basic intuition behind this theorem is as follows.

In abduction, by Corollary 2.13, if a formula F is explained, then F holds in some default extension, that is, F is satisfied by some minimal model. In circumscription, on the other hand, F should be satisfied by every minimal model, or F should hold in all default extensions. This condition is checked by computing multiple explanations of F corresponding to multiple default extensions such that those explanations cover all default extensions. Therefore, combining explanations is like an *argument system* [Poole, 1989; Poole, 1991; Helft *et al.*, 1991] consisting of two processes where one tries to find explanations of the query and the other tries to find a counter argument to refute them. In Chapter 5, a detailed analysis of this result is made to improve the efficiency and address the problem of extracting answers in previous circumscriptive theorem provers.

A more powerful version of circumscription, called *prioritized circumscription*, is defined by McCarthy [1986] to establish priorities between different kinds of abnormalities when they conflict with each other. Since the prioritized circumscription with priorities $\mathbf{P}^1 > \cdots > \mathbf{P}^k$ can be written as a conjunction of (non-prioritized) circumscriptions [Lifschitz, 1985]:

$$CIRC(T; \mathbf{P}^1 > \cdots > \mathbf{P}^k; \mathbf{Z}) \equiv \bigwedge_{i=1}^{k} CIRC(T; \mathbf{P}^i; \mathbf{P}^{i+1}, \ldots, \mathbf{P}^k, \mathbf{Z}) \,,$$

it is also equivalent to a conjunction of the corresponding ECWAs [Gelfond *et al.*, 1989]. Therefore, every circumscriptive theorem prover can be used for the prioritized case as discussed in [Przymusinski, 1989]. While prioritized circumscription performs skeptical prediction, Brewka [1989b] proposes a credulous approach to prioritization of defaults by generalizing the Theorist framework to theory formation with multiple layers:

$$(\Gamma^1, \ldots, \Gamma^k) \,,$$

where default hypotheses in Γ^i $(1 \leq i \leq k-1)$ are applied as many as possible prior to application of those in Γ^{i+1}, forming maximally consistent theories (*preferred subtheories*). A formula is predicted if there is a preferred subtheory from which the formula follows. Unlike the non-prioritized case, such a credulous prediction cannot be simply related with abductive explanations, and may take the form of an argument system as seen in [Baker and Ginsberg, 1989] so that its computation becomes similar to skeptical prediction.

As a summary of Sections 2.3.1 and 2.3.2, we now present how to perform default prediction, either credulous or skeptical, by using a theory formation framework. According to the following four steps, we can deal with commonsense knowledge in a theory formation framework to apply abduction for default prediction:

1. Each universally quantified formula, $\forall \mathbf{x}\,(H(\mathbf{x}))$, expressing a typical property is handled as a hypothesis which is separated from the reliable facts.

2. Each hypothesis is associated with a new "negative name", or an *Ab*-normal predicate [McCarthy, 1986]. For instance, a new predicate Ab_i is introduced for a formula $H_i(\mathbf{x})$ in the hypotheses, then $\neg Ab_i(\mathbf{x})$ is newly hypothesized instead of $H_i(\mathbf{x})$, and the formula

$$\forall \mathbf{x}\,(\neg Ab_i(\mathbf{x}) \supset H_i(\mathbf{x}))$$

   is added to the facts.

3. For credulous prediction, an explanation is just sought. If the goal is explainable, then there is a minimal model satisfying it.

4. For skeptical prediction, the *Ab* predicates are minimized. To do so, several explanations of the goal are computed, and a disjunction of explanations the negation of which is not explainable is sought. If such a combination is found, then every minimal model satisfies the goal.

2.3.3 Abduction and Logic Programming

So far, we have been concerned with the abductive framework (Σ, Γ) in which both Σ and Γ are sets of first-order formulas. Using this framework, nonmonotonic reasoning formalisms can be related to first-order abduction provided that the axiom set can be represented by a set Σ (or T) of first-order formulas and typical or abnormal information can be expressed by meta statements Γ, as seen in the cases of prerequisite-free normal default theories (Section 2.3.1) and circumscription (Section 2.3.2).

In this subsection, we examine the following two possible extensions of abduction:

1. Extending first-order abduction to capture nonmonotonic modal logics.

2. Expressing abduction itself within non-standard logics as "host" languages.

In particular, we focus on relationships between logic programming and abduction.

We have seen, in Section 2.3.1, an example of the first extension where *nonnormal default theories* are simulated by the Theorist framework with *constraints*. Although such a simulation is not exact with respect to the definition of default extensions, it overcomes the problem that some default theories do not have default extensions. On the other hand, we noted in Section 2.2.3 that a logic program containing *negation as failure* can be regarded as a nonnormal default theory. Here, we consider the

problem of how exactly logic programs can be simulated (and computed) by first-order abduction with some constraints.

Eshghi and Kowalski [1989] give an abductive interpretation of negation as failure in the class of *general logic programs*. Their approach is followed by [Kakas and Mancarella, 1990] and is surveyed by [Kakas *et al.*, 1992]. Suppose that we are given a general logic program Π. For each negation-as-failure formula $not\, P(\mathbf{x})$, the formula $P^*(\mathbf{x})$ is associated and is dealt with as a hypothesis, where P^* is a new predicate symbol not appearing anywhere in Π. The program Π is thereby transformed into the definite Horn program Π^*. Negation as failure is simulated by making negative conditions that can be abducible and by imposing appropriate constraints as follows. As in the Theorist framework, an atomic formula G can be predicted if there is a set E^* of new atoms (explanation) such that

1. $\Pi^* \cup E^* \models G$, and
2. $\Pi^* \cup E^*$ satisfies the following *integrity constraints*: [11]

$$\neg(\, P(\mathbf{t}) \wedge P^*(\mathbf{t})\,) \text{ and } P(\mathbf{t}) \vee P^*(\mathbf{t}) \quad \text{for every ground atomic formula } P(\mathbf{t})\,.$$

Note that when the head of a rule is empty, it means that a contradiction occurs if the body is satisfied. Therefore, the last integrity constraints can be represented by the following two schemata without heads:

$$\begin{aligned} &\leftarrow P(\mathbf{x}),\, P^*(\mathbf{x}) && \text{for every predicate symbol } P\,, \\ &\leftarrow not\, P(\mathbf{x}),\, not\, P^*(\mathbf{x}) && \text{for every predicate symbol } P\,. \end{aligned}$$

Eshghi and Kowalski [1989] show that there is a 1-1 correspondence between the answer sets (or *stable models*) of Π and the abductive extensions of Π^*. For example, if the program contains only one rule:

$$P \leftarrow not\, P\,,$$

then there is no stable model of the program. If we translate the rule into:

$$P \leftarrow P^*\,,$$

then there is no set of hypotheses satisfying the above second condition, because (1) the set $\{P^*\}$ derives P and violates the first integrity constraint, and (2) the set

[11] There are several logical specifications of integrity constraints. See [Reiter, 1990] for their differences. We simply regard integrity constraints as epistemic or meta-level statements about "what a logic program should know" or "what should be true about a logic program" [Reiter, 1990; Lifschitz, 1991].

$\emptyset$ does not derive anything and violates the second integrity constraint. Therefore, the abductive framework with integrity constraints can give the precise meaning of general logic programs.

However, difficulty arises in dealing with integrity constraints. The question is how to predict a formula without actually computing an abductive extension which satisfies the integrity constraints. Unfortunately, we cannot have any elegant top-down proof procedure which is sound with respect to the stable model semantics. In fact, Eshghi and Kowalski [1989] show an abductive proof procedure for general logic programs by incorporating consistency checks into SLD-resolution [Lloyd, 1984], but it is incorrect with respect to the above abductive interpretation of negation as failure. We will show such a counter example in Chapter 7. The reason why such a top-down approach fails is that a general logic program can be interpreted as a nonnormal default theory which does not have the following local property (*semi-monotonicity* [Reiter, 1980]): for default theories (D, W) and (D', W), if $D' \subseteq D$, then for any default extension X' of (D', W), there is a default extension X of (D, W) such that $X' \subseteq X$. Guerreiro *et al.* [1990] show weaker necessary conditions for the existence of a sound and complete proof procedure for default theories, and a modified definition of default extensions for which a proof procedure exists. Also, [Kakas *et al.*, 1992] summarizes more recent work on modification of the stable model semantics for which a top-down proof procedure works. In contrast, we have shown in Sections 2.3.1 and 2.3.2 that query-answering in both credulous and skeptical default prediction based on first-order logic can be computed by finding (combinations of) explanations of the query without computing any default extensions. This is because the class of normal default theories satisfies the semi-monotonicity [Reiter, 1980, Theorem 3.2].

In Chapter 7, we will show a bottom-up procedure for every class of logic program and disjunctive database within the answer set semantics [Inoue *et al.*, 1992a]. Instead of using the above abductive characterization, we hypothesize the epistemic statement about P to evaluate the negation-as-failure formula $not\, P$. That is, we assume that P *should* (or *should not*) hold in the fixpoint. Moreover, we deal with integrity constraints as a part of the program instead of separating them from the program. Thus, this kind of abduction is beyond first-order logic, while we will use a first-order model generation theorem prover.

For the second extension of abduction, we now briefly introduce recent work on abduction within non-standard logics. Again, those proposals can be found in various extensions of logic programming. Kakas and Mancarella [1990] propose *abductive logic programming* which supplies abduction not only for negation as failure but in a general way. Their abductive framework is defined by a triple $\langle\, \Pi, \Gamma, I \,\rangle$, where Π is a general logic program, Γ is a pre-specified set of atomic formulas (hypotheses), and I is a set of integrity constraints. Then, a *generalized stable model* of $\langle\, \Pi, \Gamma, I \,\rangle$

is defined as a stable model of $\Pi \cup E$ which satisfies I, where E is any subset of ground instances of elements of Γ. Abductive logic programming is an extension of both non-abductive logic programming and first-order Horn abduction in the sense that the logic is non-standard as the axioms are represented by using the negation-as-failure operator (not) and the non-classical connective ($\leftarrow$). In this framework, an atomic formula G is defined to be predicted if there is a generalized stable model satisfying G. Thus, their prediction is a credulous approach. On the other hand, Gelfond [1990] proposes a skeptical framework within extended disjunctive databases that allow disjunctions in heads and classical negation as well as negation as failure.

For $\langle \Pi, \Gamma, I \rangle$, Kakas and Mancarella [1990] further transform the negation-as-failure formulas according to the abductive characterization by [Eshghi and Kowalski, 1989], and show a 1-1 correspondence between the generalized stable models of $\langle \Pi, \Gamma, I \rangle$ and the abductive extensions of the transformed framework $\langle \Pi^*, \Gamma \cup \Gamma^*, I \cup I^* \rangle$, where Π^*, Γ^* and I^* are obtained by the transformation described before. In this way, abductive logic programming can be reduced to first-order abduction with integrity constraints. However, this transformation inherits the difficulty of computation from Eshghi and Kowalski's abductive interpretation of negation as failure. In fact, the proof procedure for abductive logic programming is defined as an extension of Eshghi and Kowalski's proof procedure in [Kakas *et al.*, 1992], and suffers from the same soundness problem that was described earlier. Alternatively, [Inoue, 1991a] and [Satoh and Iwayama, 1991] show that the abductive framework $\langle \Pi, \Gamma, I \rangle$ can be transformed into a single general logic program. For each atomic formula $P(\mathbf{x})$ in Γ, they introduce the negative literal $\neg P(\mathbf{x})$ and a pair of new rules: [12]

$$P(\mathbf{x}) \leftarrow not\ \neg P(\mathbf{x}),$$
$$\neg P(\mathbf{x}) \leftarrow not\ P(\mathbf{x}).$$

Also, each integrity constraint in I can be transformed into a rule without head and thus can also be embedded in the program. Then, there is a 1-1 correspondence between the generalized stable models of $\langle \Pi, \Gamma, I \rangle$ and the stable models of the transformed program. In this transformation, again, we cannot have a top-down proof procedure which is sound with respect to the (generalized) stable model semantics because we remain in the framework of general logic programs. Thus, [Satoh and Iwayama, 1991] and [Inoue *et al.*, 1992a] propose bottom-up procedures for computation of (generalized) stable models.

Notice that, in the above simulation of hypotheses by negation as failure, *classical negation* ($\neg$) is employed. This fact implies that we do not have to restrict ourselves

[12] Satoh and Iwayama do not use the negative literal $\neg P$ but use the new predicate symbol $\tilde{P}$ for each predicate symbol P appearing in Γ because they do not allow for classical negation.

to remaining in general logic programs. Namely, abductive logic programming can be defined within the class of *extended logic programs* [Gelfond and Lifschitz, 1990]. In fact, Inoue [1991a] proposes a much more general framework for hypothetical reasoning, called a *knowledge system* (see Chapter 6). A knowledge system is defined as a pair (Π, Γ), where both Π and Γ are extended logic programs. The semantics of (Π, Γ) is given by the answer sets of $\Pi \cup E$ for every $E \subseteq \Gamma$ such that $\Pi \cup E$ is consistent. This framework can be broadly applied to default reasoning, abduction, closed world assumption, and inconsistency resolution. In contrast, the framework of [Kakas and Mancarella, 1990] can be used only for abduction. We will see, in Chapter 6, that every knowledge system can be transformed into an extended (and then a general) logic program.

Apart from logic programming, there are some proposals for hypothetical reasoning based on non-standard logics. Siegel and Schwind [1991] propose a *hypothesis theory* based on *modal logic* **T**. Siegel and Schwind also show a translation of a nonnormal default theory into a hypothesis theory. Although their definition makes every hypothesis theory always have some kind of extension, to see that such an extension is really a default extension of the default theory, we need to check if it satisfies the constraint that the "hypothesized" and "believed" formulas coincide. This kind of extra check is also employed by other characterizations of default logic, such as [Lin, 1991; Lifschitz, 1991]. In Chapter 7, we will see that for logic programs, i.e., a special important class of nonnormal default theories, this check can be carried out very simply by the constraint that the "assumed" literals are actually "derived".

Lots of effort has been made into the computation of nonmonotonic modal logics including logic programs and disjunctive databases over the last few years, but more investigation is still needed. While we cannot expect the existence of purely top-down proof procedures for these logics that rely on explanation-finding mechanisms within classical logic, it is an open question how much we can incorporate the goal information into bottom-up procedures so that the search of the proof is as localized as possible. We should also characterize the largest classes of abductive and nonmonotonic logics that have proof procedures both syntactically and semantically.

2.3.4 Summary

In this chapter, we have seen various relationships between abduction and nonmonotonic reasoning. A summary follows:

1. Abduction is intrinsically nonmonotonic inference, because it is non-deductive.

2. Abduction can be formalized by the simplest class of default logic, that is, prerequisite-free normal default theories. This class of default theories fits in

with the concept of theory formation, and has many good computational properties. Thus, abduction can be computed using (and modifying) efficient theorem proving procedures for deduction.

3. Since default logic admits multiple extensions as possible sets of beliefs, there are two approaches, credulous and skeptical, to default prediction. Both approaches can be handled using abduction. Thus, circumscription can be implemented by utilizing abduction.

4. For more general default theories, first-order abduction cannot be used directly for computation, though it provides an approximation with some constraints.

5. Logic programming and abduction are also closely related. Logic programs belong to a class of nonnormal default theories, and correct computation can be carried out in a bottom-up manner.

6. Therefore, abduction can play a key role in the process of advanced (and non-deductive) inference by filling the gap between deduction and non-deduction.

Some of the main ideas presented in this chapter will be covered in detail by the following chapters of this dissertation. In particular, we shall address the following problems:

1. We need a complete characterization of abduction which enables efficient deductive theorem proving techniques to be utilized for computing explanations. We show, in Chapter 3, that abduction is characterized very elegantly by deduction as a consequence-finding problem. In Chapter 4, one of the most promising theorem proving techniques is provided for computing abduction.

2. We need an efficient query answering mechanism for skeptical prediction, in particular for circumscription. This is investigated in Chapter 5, where the techniques introduced in Chapters 3 and 4 are applied.

3. We need a representational methodology and a computational mechanism for more general default theories including logic programs. For these purposes, we propose theory formation within extended logic programs in Chapter 6, and consider its bottom-up computation with epistemic hypotheses in Chapter 7.

In this chapter, we have also pointed out a number of important problems which have not been solved yet or should be pursued further. In Chapter 8, we shall consider some of these issues.

Chapter 3

Characterization of Consequence-Finding

In this and the next chapters, we study proof-theoretic properties of abduction and nonmonotonic reasoning. In this chapter, we re-evaluate the consequence-finding problem within first-order logic, which is a key problem for abduction. Consequence-finding will be generalized to the problem in which only interesting clauses having a certain property (called characteristic clauses) should be found. This chapter also shows that the use of characteristic clauses enables characterization of various reasoning problems of interest to AI. A proof procedure based on the results of this chapter will be shown in the next chapter. The importance of the results presented lies in their applicability to a wide class of AI problems such as procedures for abduction, nonmonotonic reasoning, prime implicates, truth maintenance systems and constraint satisfaction, and their possible applications including diagnosis, design and planning.

3.1 Consequence-Finding Problem

It is well known in automated deduction that while resolution [Robinson, 1965] is complete for proof-finding (*refutation* complete), that is, it can derive $false$ from every unsatisfiable set of formulas, it is not deductively complete for finding every logical consequence of a given satisfiable set of formulas. For example, resolution cannot derive the formula $Q \vee R$ from a set of formulas $\Sigma = \{P, \neg P \vee Q\}$ although $\Sigma \models Q \vee R$. Lee [1967] addresses himself to this problem and defines the *consequence-finding* problem, which we shall express in the following form:

Given a set of formulas Σ and a resolution procedure **RP**, for any logical consequence T of Σ, can **RP** derive a logical consequence S of Σ such that S subsumes T?

If a resolution procedure is complete for consequence-finding, then it is useful in spite of lacking deductive completeness because in general the logical consequences not derivable from the axiom set are neither interesting nor useful. Namely, such a formula is subsumed by some formula derivable from the axiom set and thus it is weak and redundant. In the above example, the resolution principle can derive Q from $\Sigma = \{P, \neg P \vee Q\}$. Since Q subsumes $Q \vee R$, we do not have to mind if the meaningless theorem $Q \vee R$ cannot be actually derived. In other words, a resolution procedure that is complete for consequence-finding can derive all the meaningful theorems of the axiom set.

Historically, consequence-finding had been investigated intensively ever since the resolution principle was invented by Robinson [1965] for proof-finding. Lee's completeness theorem [1967] was proven for the original resolution principle. Slagle, Chang and Lee [1969] extended the result to various kinds of semantic resolution (including I-semantic resolution, hyperresolution, and PI-resolution). However, once Minicozzi and Reiter [1972] extended these results to some *linear resolution* strategies in the early 70s, consequence-finding was abandoned in the research of automated theorem proving and attention was directed towards proof-finding only. [1] It appears that there are three reasons for this discouragement:

1. The results presented by [Minicozzi and Reiter, 1972] are in some sense negative.

   - Linear resolution involving *A-ordering* (a total ordering of all the ground instances of the atomic formulas of the axiom set [Kowalski and Hayes, 1969; Reiter, 1971]) is complete for consequence-finding only if all possible A-orderings are tried when seeking to generate all possible target theorems. This result is far from being a practical strategy.
   - Linear resolution involving *C-ordering* (literals are ordered in each clause in the axiom set [Loveland, 1978; Reiter, 1971]), such as Model Elimination [Loveland, 1969; Loveland, 1978] and its variants [Kowalski and Kuhner, 1971; Chang and Lee, 1973; Shostak, 1976], may be the most familiar and efficient linear input procedures complete for refutation in first-order

[1] One can see that textbooks on resolution-based theorem proving, such as [Chang and Lee, 1973; Loveland, 1978], have no sections on consequence-finding.

logic. [2] This is because they contain many restriction strategies. They are, however, incomplete for consequence-finding. Thus, the completeness result that we would most like to obtain does not hold.

2. Even if a resolution procedure is complete for consequence-finding, it is in general neither practical nor useful to find all the theorems of the axiom set. Previous methods first generate all derivable theorems by using consequence-finding procedures, and then filter them with some given criteria. There has not been an intelligent method that directly searches for only interesting consequences without using "generate-and-test" manners.

3. As opposed to proof-finding, which can be used, for instance, in planning and synthesis problems where answer extraction techniques are available to obtain useful information, consequence-finding has lacked useful applications in AI.

In this and the next chapters, however, we will argue that the above three problems can be solved well.

Before we investigate the first and the second problems (in the next chapter), we will give new perspectives to the consequence-finding problem in this chapter. In particular, we will show a solution to the third problem, that is, the usefulness of consequence-finding in AI. This goal will be achieved via the following two subgoals:

1. We generalize consequence-finding to the problem in which only only interesting clauses that are subject to user-supported constraints should be found. We call such restricted consequences *characteristic clauses*. Also, when a new axiom is added to the theory, characteristic clauses newly derived from the new axiom set are called *new characteristic clauses*.

2. We show how various AI problems can be stated by using this notion of (new) characteristic clauses. For example,

   (a) Resolution-based procedures for *abduction* such as [Pople, 1973; Cox and Pietrzykowski, 1986; Finger, 1987; Poole *et al.*, 1987; Stickel, 1989] generate explanations of queries and actually produce some formulas other than the empty clause. In other words, such abductive procedures can utilize consequence-finding procedures.

[2] Technically, all Model Elimination variants using the information of literals resolved upon are not formally called resolution procedures because (single-sorted) resolution has no linear input restriction complete for refutation in first-order logic. Here, we take the liberty of calling all Model Elimination variants *linear resolution* procedures.

(b) In the propositional case, the consequence-finding problem is reduced to the problem of finding the *prime implicants/implicates* [Quine, 1952; Tison, 1967]. An algorithm to compute prime implicates can be utilized for the *clause management system* (CMS) [Reiter and de Kleer, 1987] that is a generalization of the ATMS [de Kleer, 1986].

(c) Query answering for *circumscription* [McCarthy, 1980; Lifschitz, 1985] and for databases augmented with the *closed-world assumption* [Bossu and Siegel, 1985; Gelfond *et al.*, 1989; Minker and Rajasekar, 1990] can utilize an abductive procedure [Inoue and Helft, 1990; Helft *et al.*, 1991] and thus can be implemented by a consequence-finding procedure.

Consequently, the importance of the results we present in this chapter lies in their applicability to a wide class of AI problems, in particular to applications of abductive and nonmonotonic reasoning that include diagnosis, design, and planning. In other words, the methods may contribute to a better understanding of many AI techniques as well as better implementation of them.

3.2 Characterizing Logical Consequences

This section presents basic formalism of logical consequences of a theory within first-order logic. We begin with some definitions and notations, most of which are adopted from [Chang and Lee, 1973; Loveland, 1978], as follows.

We define a *theory* as a set of clauses. [3] A *clause* is a disjunction of *literals*, each of which is a possibly negated atomic formula, with no literal repeated. Each variable in a clause is assumed to be universally quantified. The *length* of a clause is the number of literals it contains. The *empty clause* is the clause of length 0 and is denoted as $\Box$. Set operations of clauses are defined in such a way that each clause is identified with the set of its literals: If C and D are two clauses, $C - D$ denotes a clause whose literals are those in the difference of C and D; $C \cup D$ ($C \cap D$) denotes a clause whose literals are those in the union (intersection) of C and D; $C \subseteq D$ means that every literal in C occurs in D. For example, when $C_1 = P \vee Q \vee R$ and $C_2 = Q \vee S$, $C_1 - C_2 = P \vee R$, $C_1 \cup C_2 = P \vee Q \vee R \vee S$ and $C_1 \cap C_2 = Q$. If Σ is a set of clauses, we mean by $\overline{\Sigma}$ the set formed by taking the negation of each clause in Σ. For example, when $\Sigma = \{ P, \neg Q \vee R \}$, $\overline{\Sigma} = \{ \neg P, Q \wedge \neg R \}$. A *conjunctive normal form (CNF) formula* is a conjunction of clauses. A formula F is *valid* if $\models F$. When k is a literal, we will use $\neg k$ to mean the complement of k. For example, if k is a

[3] Traditionally, a theory is a collection of formulas together with all the consequences of those formulas. Here, however, a *theory* denotes a *(proper) axiom set* only.

negative literal $\neg P$, then $\neg k$ is P. A valid clause is a *tautology* if it contains a pair of complimentary literals, k and $\neg k$. A clause C is a *theorem*, or a *(logical) consequence* of a theory Σ if $\Sigma \models C$. The set of theorems of Σ is denoted by $Th(\Sigma)$.

We use the following *subsumption* criterion. A *substitution* is a finite set of variables associated with terms to replace those variables, with no variable repeated. Let C and D be two clauses. C *subsumes* D if there is a substitution θ such that $C\theta \subseteq D$ and C has no more literals than D. [4] C *properly subsumes* D if C subsumes D but D does not subsume C. C *subsumes* D *with the empty substitution* if $C \subseteq D$. For a set of clauses Σ, $\mu\Sigma$ or $\mu[\Sigma]$ denotes the set of clauses in Σ not properly subsumed by any clause in Σ. Therefore, for two clauses C and D in $\mu\Sigma$, if C is subsumed by D, then C also subsumes D. Notice that $\mu[\mu\Sigma] = \mu\Sigma$. Σ is *closed under subsumption* if it satisfies $\Sigma = \mu\Sigma$. For example, when $\Sigma = \{ P \vee Q, \neg P \vee Q, \neg P \vee \neg Q \}$, both Σ and $\mu Th(\Sigma) = \{ \neg P, Q \}$ are closed under subsumption.

3.2.1 Characteristic Clauses

We use the notion of *characteristic clauses*. This is a generalized notion of logical consequences and helps to understand the computational aspects of many AI problems. The idea of characteristic clauses was introduced by Bossu and Siegel [1985] for evaluating a kind of closed-world reasoning, where the set of positive clauses derivable from a particular class of theory (groundable theory) should be found. Their idea was later generalized by Siegel [1987] for propositional logic. Inoue [1990b] investigated its properties extensively. The description below is more general than [Bossu and Siegel, 1985; Siegel, 1987; Inoue, 1990b] in the sense that it is not limited to closed-world reasoning and that it deals with the general (i.e., first-order) case instead of just the propositional case. Also, the notion of characteristic clauses is independent of implementation; we do not assume any particular resolution procedure in this and the next sections. Informally speaking, characteristic clauses are intended to represent

[4] This definition of subsumption is called *θ-subsumption* in [Loveland, 1978]. Unlike the propositional case, we need the second condition that the length of C does not exceed the length of D. Notice that a clause C and its factor D always satisfy the first condition (i.e., $\exists\theta$ such that $C\theta = D$) but C has more literals than D. The second condition enables a production field (see Definition 3.1 (2)) with the restriction on the maximum length of clauses (see Example 3.2 (3)) to be stable (see Definition 3.1 (5)). For example, let $C_1 = \neg P(x,y) \vee \neg P(y,z) \vee Q(f(x,z))$ and $D_1 = \neg P(x,x) \vee Q(f(x,x))$. Although D_1 can be obtained from C_1 through factoring, we do not say that C_1 subsumes D_1. For another example, let $C_2 = P(x,x)$ and $D_2 = P(x,y) \vee P(z,z)$. Although $C_2 \equiv D_2$ is valid, C_2 properly subsumes D_2. Note also that, if the subsumption criterion was described without the second condition, then for a theorem D of Σ, there might be no theorem C of Σ not properly subsumed by any theorem of Σ such that C subsumes D, thus Proposition 3.5 would not hold.

"interesting" clauses for a given problem, and are constructed over a sub-vocabulary of the representation language called a *production field*.

Definition 3.1 (1) Let $\mathcal{R}$ be the set of all predicate symbols in the language. For $\mathbf{R} \subseteq \mathcal{R}$, we denote by $\mathbf{R}^+$ ($\mathbf{R}^-$) the positive (negative) occurrences of predicates from $\mathbf{R}$ in the language. The set of all atomic formulas $\mathcal{R}^+$ is denoted as $\mathcal{A}$, and the set of literals $\mathcal{R}^+ \cup \mathcal{R}^-$ is denoted as $\mathcal{L}$ ($= \mathcal{A} \cup \overline{\mathcal{A}}$). A set of literals $\mathbf{L} \subseteq \mathcal{L}$ is *closed under instantiation* if, a literal containing variables is in $\mathbf{L}$ only if all its instances are also in $\mathbf{L}$. Note that, $\mathbf{L}$ is closed under instantiation if and only if $\mathbf{L}$ can be written in the form of ${\mathbf{R}_1}^+ \cup {\mathbf{R}_2}^-$, where $\mathbf{R}_1 \subseteq \mathcal{R}$ and $\mathbf{R}_2 \subseteq \mathcal{R}$.
(2) A *production field* $\mathcal{P}$ is represented by a pair, $\langle\, \mathbf{L}, Cond \,\rangle$, where $\mathbf{L}$ (called the *characteristic literals* of $\mathcal{P}$) is a subset of $\mathcal{L}$ and is closed under instantiation, and $Cond$ is a certain condition to be satisfied. When $Cond$ is not specified, $\mathcal{P}$ is just denoted as $\langle\, \mathbf{L} \,\rangle$. The production field $\langle\, \mathcal{L} \,\rangle$ is denoted as $\mathcal{P}_{\mathcal{L}}$.
(3) A clause C is said to *belong to a production field* $\mathcal{P} = \langle\, \mathbf{L}, Cond \,\rangle$ if every literal in C belongs to $\mathbf{L}$ and C satisfies $Cond$.
(4) Let Σ be a theory. The set of theorems of Σ belonging to $\mathcal{P}$ is denoted as $Th_{\mathcal{P}}(\Sigma)$.
(5) A production field $\mathcal{P}$ is *stable* if for any two clauses C and D such that C subsumes D, D belongs to $\mathcal{P}$ only if C belongs to $\mathcal{P}$.

Example 3.2 The following are examples of stable production fields.
(1) $\mathcal{P}_1 = \mathcal{P}_{\mathcal{L}}$: $Th_{\mathcal{P}_1}(\Sigma)$ is equivalent to $Th(\Sigma)$.
(2) $\mathcal{P}_2 = \langle\, \mathcal{A} \,\rangle$: $Th_{\mathcal{P}_2}(\Sigma)$ is the set of positive clauses implied by Σ.
(3) $\mathcal{P}_3 = \langle\, \overline{\Gamma},$ *length is fewer than* $k \,\rangle$ where $\Gamma \subseteq \mathcal{A}$: $Th_{\mathcal{P}_3}(\Sigma)$ is the set of negative clauses implied by Σ consisting of fewer than k literals in $\overline{\Gamma}$.

Example 3.3 The production field $\mathcal{P}_4 = \langle\, \Gamma,$ *length is more than* $k \,\rangle$ where $\Gamma \subseteq \mathcal{L}$ is not stable. For example, if $k = 2$ and $\Gamma = \{P\}^- \cup \{Q\}^+$, then $C = \neg P(x) \vee Q(x)$ subsumes $D = \neg P(x) \vee \neg P(f(x)) \vee Q(x)$, and D belongs to $\mathcal{P}_4$ while C does not.

Definition 3.4 (Characteristic Clauses) Let Σ be a set of clauses, and $\mathcal{P}$ a production field. The *characteristic clauses of* Σ *with respect to* $\mathcal{P}$ are:

$$Carc(\Sigma, \mathcal{P}) = \mu\, Th_{\mathcal{P}}(\Sigma)\,.$$

$Carc(\Sigma, \mathcal{P})$ is constructed from the theorems of Σ that belong to a production field $\mathcal{P}$ and is closed under subsumption. Now, the problem of finding $Carc(\Sigma, \mathcal{P})$ is a generalization of the ordinary consequence-finding problem. To verify this claim, let $\mathcal{P}$ be $\mathcal{P}_{\mathcal{L}}$. From the definition of consequence-finding, for any clause $D \in Th(\Sigma)$, a complete procedure **RP** can derive a clause $C \in Th(\Sigma)$ such that C subsumes D.

Therefore, for every clause $C' \in \mu Th(\Sigma)$, **RP** can derive a clause $C'' \in \mu Th(\Sigma)$ such that C' and C'' subsume each other. Hence, we can consider that $Carc(\Sigma, \mathcal{P}_{\mathcal{L}}) = \mu Th(\Sigma)$ is contained in the set of theorems derivable from Σ by using **RP**. Note also that the empty clause $\square$ belongs to every stable production field $\mathcal{P}$, and that if Σ is unsatisfiable then $Carc(\Sigma, \mathcal{P})$ contains only $\square$. This means that proof-finding is a special case of consequence-finding.

Proposition 3.5 Let Σ be a theory, $\mathcal{P}$ a stable production field and D a clause belonging to $\mathcal{P}$. $\Sigma \models D$ if and only if there is a clause C in $Carc(\Sigma, \mathcal{P})$ such that C subsumes D. In particular, Σ is unsatisfiable if and only if $Carc(\Sigma, \mathcal{P}) = \{\square\}$.

Proof: Let D be a clause belonging to $\mathcal{P}$, and C a clause in $Carc(\Sigma, \mathcal{P})$. If C subsumes D, obviously $D \in Th_{\mathcal{P}}(\Sigma)$. Conversely, suppose that $D \in Th_{\mathcal{P}}(\Sigma)$. We prove the first claim by induction on the length $m \geq 0$ of D.

Induction basis ($m = 0$). Since $D = \square$, Σ is unsatisfiable. Since $\square$ is not subsumed by any other theorem of Σ, it is contained in $Carc(\Sigma, \mathcal{P})$. Furthermore, $Carc(\Sigma, \mathcal{P})$ does not contain any other clause so that the second claim is also proven.

Induction step. Suppose that the claim holds for clauses of maximum length $m \geq 0$ and D is a clause of length $m + 1$ in $Th_{\mathcal{P}}(\Sigma)$. If D is not properly subsumed by any clause in $Th_{\mathcal{P}}(\Sigma)$, then the proof is completed as D is in $Carc(\Sigma, \mathcal{P})$. Let C' be a clause in $Th_{\mathcal{P}}(\Sigma)$ such that C' properly subsumes D. Two cases arises:

1. C' has fewer literals than D. Then, the length of C' is not more than m. By the induction hypothesis, there is a clause $C \in Carc(\Sigma, \mathcal{P})$ such that C subsumes C'. Since C' subsumes D, C subsumes D.
2. The length of C' is the same as the length of D. Since C' subsumes D, there is a substitution θ such that $C'\theta \subseteq D$. Since $C'\theta$ belongs to $\mathcal{P}$ and is an instance of C', $C'\theta$ belongs to $Th_{\mathcal{P}}(\Sigma)$. There are two cases:
   (a) $C'\theta \subset D$. Then, the length of $C'\theta$ is not more than m. By induction hypothesis, there is a clause $C \in Carc(\Sigma, \mathcal{P})$ such that C subsumes $C'\theta$. Since $C'\theta$ subsumes D with the empty substitution, C subsumes D.
   (b) $C'\theta = D$. Then, D is an instance of C'. However, D is not an alphabetic variant of C' because D does not subsume C'.

Now, suppose that there is no clause C in $Carc(\Sigma, \mathcal{P})$ such that C subsumes C'. Then, by the above discussion, there must be an infinite sequence of clauses belonging to $Th_{\mathcal{P}}(\Sigma)$:

$$C_0(= D),\ C_1(= C'),\ C_2,\ \ldots,\ C_{i-1},\ C_i,\ \ldots\ ,$$

such that for all C_i $(i \geq 1)$, the length of C_i is the same as the length of D and that C_{i-1} is an instance of C_i but is neither an alphabetic variant of C_i nor C_i itself. However, since the number of terms appearing in D is finite, no C_i can contain an infinite number of variables. Therefore, there must be, in the sequence, C_j and C_{j-1} $(j \geq 1)$ which cannot satisfy such conditions, contradiction. □

3.2.2 New Characteristic Clauses

As we will see later, when new information is added to the theory, it is often necessary to compute the newly derivable consequences caused by this new information. For this purpose, consequence-finding is extended to look for such *ramification* [Finger, 1987] of new information.

Definition 3.6 (New Characteristic Clauses) Let Σ be a set of clauses, $\mathcal{P}$ a production field, and F a formula. The *new characteristic clauses of* F *with respect to* Σ *and* $\mathcal{P}$ are:

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th(\Sigma)\,]\,.$$

In other words, $C \in Newcarc(\Sigma, F, \mathcal{P})$ if and only if C satisfies the conditions:

1. $\Sigma \cup \{F\} \models C$,
2. C belongs to $\mathcal{P}$,
3. $\Sigma \not\models C$, and
4. If a clause C' subsumes C and satisfies the above three, then C' is subsumed by C.

Both the characteristic clauses and the new characteristic clauses specify what kind of clauses should be computed as (new) theorems. The next three propositions show the connections between the characteristic clauses and the new characteristic clauses. Firstly, $Newcarc(\Sigma, F, \mathcal{P})$ can be represented by the set difference of two sets of characteristic clauses.

Proposition 3.7 A clause is a new characteristic clause of F with respect to Σ and $\mathcal{P}$ if and only if it is a characteristic clause of $\Sigma \cup \{F\}$ but is not a characteristic clause of Σ. Namely,

$$Newcarc(\Sigma, F, \mathcal{P}) = Carc(\,\Sigma \cup \{F\},\, \mathcal{P}) - Carc(\Sigma,\, \mathcal{P})\,.$$

Proof: By Definition 3.6, $Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th(\Sigma)\,]$. Since $Th_{\mathcal{P}}(\Sigma \cup \{F\})$ contains only clauses belonging to $\mathcal{P}$, the definition can be rewritten as:

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,Th_{\mathcal{P}}(\Sigma \cup \{F\}) - Th_{\mathcal{P}}(\Sigma)\,]\,.$$

Now, let $X = Th_{\mathcal{P}}(\Sigma \cup \{F\})$ and $Y = Th_{\mathcal{P}}(\Sigma)$. Notice that $Y \subseteq X$. We will prove that $\mu[X - Y] = \mu X - \mu Y$.

Let C be a clause belonging to $\mathcal{P}$ such that $C \in \mu[X - Y]$. Then obviously $C \in X - Y$ and thus $C \in X$. Now assume that $C \notin \mu X$. Then there exists a clause D in μX such that D properly subsumes C. Since C is not properly subsumed by any clause in $X - Y$, it holds that $D \in Y$. By the fact that D subsumes C, it holds that $C \in Y$, contradiction. Therefore, $C \in \mu X$. Clearly, by $C \notin Y$, $C \notin \mu Y$. Hence, $C \in \mu X - \mu Y$.

Conversely, assume that $C \in \mu X - \mu Y$. Firstly, we must prove that $C \in X - Y$. Suppose to the contrary that $C \in Y$. Since $C \notin \mu Y$, there exists in μY a clause D such that D properly subsumes C. However, as $Y \subseteq X$, $D \in X$ holds, thus contradicting the fact that $C \in \mu X$. Therefore, $C \in X - Y$. Now, assume that $C \notin \mu[X - Y]$. Then, there exists in $X - Y$ a clause D' such that D' properly subsumes C, again contradicting the fact that $C \in \mu X$. Hence, $C \in \mu[X - Y]$. □

When F is a CNF formula, $Newcarc(\Sigma, F, \mathcal{P})$ can be decomposed into a series of *primitive Newcarc operations* each added new formula of which is just a single clause.

Proposition 3.8 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\,\bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})\,]\,,$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$.

Proof: Notice that $X - Z = (X - Y) \cup (Y - Z)$ holds for sets X, Y and Z such that $Z \subseteq Y \subseteq X$. Also, for two sets of clauses, X and Y, $\mu[\, X \cup Y \,] = \mu[\, \mu X \cup \mu Y \,]$ holds. Therefore,

$$
\begin{aligned}
& Newcarc(\Sigma, F, \mathcal{P}) \\
= \;& \mu\,[\, Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th(\Sigma) \,] \\
= \;& \mu\,[\, Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th_{\mathcal{P}}(\Sigma) \,] \\
= \;& \mu\,[\, (\, Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_m\}) - Th_{\mathcal{P}}(\Sigma \cup \{C_1, \ldots, C_{m-1}\})) \\
& \quad \cup \cdots \cup (\, Th_{\mathcal{P}}(\Sigma \cup \{C_1\}) - Th_{\mathcal{P}}(\Sigma)\,) \,] \\
= \;& \mu\,[\, (\, Th_{\mathcal{P}}(\Sigma_{m+1}) - Th_{\mathcal{P}}(\Sigma_m)\,) \cup \cdots \cup (\, Th_{\mathcal{P}}(\Sigma_2) - Th_{\mathcal{P}}(\Sigma_1)\,) \,] \\
= \;& \mu\,[\, \mu[\, Th_{\mathcal{P}}(\Sigma_{m+1}) - Th_{\mathcal{P}}(\Sigma_m) \,] \cup \cdots \cup \mu[\, Th_{\mathcal{P}}(\Sigma_2) - Th_{\mathcal{P}}(\Sigma_1) \,] \,] \\
= \;& \mu\,[\, Newcarc(\Sigma_m, C_m, \mathcal{P}) \cup \cdots \cup Newcarc(\Sigma_1, C_1, \mathcal{P}) \,] \\
= \;& \mu\,[\, \bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P}) \,] .
\end{aligned}
$$

□

Finally, the characteristic clauses $Carc(\Sigma, \mathcal{P})$ can be constructively expressed by using primitive $Newcarc$ operations. Now, let $Taut(\mathbf{L})$ (the tautologies defined over the literals of $\mathbf{L}$) be the set of valid clauses consisting of two literals from $\mathbf{L} \subseteq \mathcal{L}$.

Proposition 3.9 (Incremental Construction of the Characteristic Clauses) Let Σ be a set of clauses, C a clause and $\mathcal{P} = \langle\, \mathbf{L}, Cond \,\rangle$ a stable production field.

$$
\begin{aligned}
Carc(\emptyset, \mathcal{P}) &= \mu\,\{\; T \mid T \in Taut(\mathbf{L}) \text{ and } T \text{ satisfies } Cond \,\}, \\
Carc(\Sigma \cup \{C\}, \mathcal{P}) &= \mu\,[\, Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P}) \,] .
\end{aligned}
$$

Proof: The first equation follows immediately from Definition 3.4. Now,

$$
\begin{aligned}
& Carc(\Sigma \cup \{C\}, \mathcal{P}) \\
= \;& \mu\, Th_{\mathcal{P}}(\Sigma \cup \{C\}) \\
= \;& \mu\,[\, Th_{\mathcal{P}}(\Sigma \cup \{C\}) \cup Th_{\mathcal{P}}(\Sigma) \,] \\
= \;& \mu\,[\, \mu Th_{\mathcal{P}}(\Sigma \cup \{C\}) \cup \mu Th_{\mathcal{P}}(\Sigma) \,] \\
= \;& \mu\,[\, Carc(\Sigma, \mathcal{P}) \cup Carc(\Sigma \cup \{C\}, \mathcal{P}) \,] \\
= \;& \mu\,[\, Carc(\Sigma, \mathcal{P}) \cup (\, Carc(\Sigma \cup \{C\}, \mathcal{P}) - Carc(\Sigma, \mathcal{P})\,) \,] \\
= \;& \mu\,[\, Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P}) \,] \;\text{ (by Proposition 3.7)}
\end{aligned}
$$

□

Implementation of computation of these consequences depends heavily on which operation between $Carc$ and $Newcarc$ is chosen as the basis: $Carc$ can be taken up as the basic operation in Proposition 3.7, while primitive $Newcarc$ can be used for Propositions 3.8 and 3.9. The former approach can be seen in Bossu and Siegel's [1985] saturation procedure. To derive the new positive theorems, their method should first deduce all the $Carc(\Sigma, \mathcal{P})$ prior to giving $Carc(\Sigma \cup \{F\}, \mathcal{P})$, where the characteristic literals of $\mathcal{P}$ are fixed to the ground atoms (see Example 3.2 (2)). On the other hand, Siegel [1987] demonstrates that the latter approach outperforms the former to compute the new characteristic clauses of a propositional theory. Recall that our definitions of the (new) characteristic clauses are more general than those of [Bossu and Siegel, 1985; Siegel, 1987]. We will discuss this issue later in Section 4.2.2.

3.3 Applications

We illustrate how the use of the (new) characteristic clauses enables an elegant definition and precise understanding of many AI problems.

3.3.1 Abduction

There are many proposals for a logical account of *abduction* [Pople, 1973; Cox and Pietrzykowski, 1986; Finger, 1987; Poole *et al.*, 1987; Poole, 1989; Stickel, 1989; Demolombe and Fariñas del Cerro, 1991], whose task is to generate explanations of a query.

Definition 3.10 Let Σ be a theory, $\Gamma \subseteq \mathcal{L}$ and G a closed formula.
(1) An *abductive framework* is a pair (Σ, Γ). In this case, Γ is called the *hypotheses* of the abductive framework, and is assumed to be closed under instantiation.
(2) A formula E is an *explanation of* G *from* (Σ, Γ) if

1. E is a conjunction of literals of Γ with no literal repeated,
2. $\Sigma \cup \{ E \} \models G$, and
3. $\Sigma \cup \{ E \}$ is consistent.

(3) Let E and E' be two explanations of G from (Σ, Γ). E is a *subexplanation* of E' if there is a substitution θ such that every literal in $E\theta$ occurs in E' and E has no more literals than E'. E is a *proper subexplanation* of E' if E is a subexplanation of E' but E' is not a subexplanation of E.
(4) An explanation E of G is *minimal* if there is no proper subexplanation of E.

(5) An explanation of G is *ground* if it is a conjunction of ground literals.
(6) An *extension of* (Σ, Γ) is the set of logical consequences of $\Sigma \cup \mathbf{M}$ where $\mathbf{M}$ is a maximal set of ground elements of Γ such that $\Sigma \cup \mathbf{M}$ is consistent.

Note that Definition 3.10 is based on an abductive reasoning system Theorist [Poole *et al.*, 1987; Poole, 1988; Poole, 1989] except that Theorist deals with ground explanations only. Note that each free variable in an explanation can be assumed to be existentially quantified. It turns out that allowing variables to appear in explanations enables characterization of abduction more naturally (Proposition 3.11). For example, let $\Sigma = \{ \neg P(x) \vee Q(x) \}$, $\Gamma = \{P\}^+$, and $G = \exists x\,(Q(x))$. Then, $P(x)$ is an explanation of G from (Σ, Γ) and is the only minimal explanation of G (we identify with it all its alphabetic variants). If abduction is restricted to accepting only ground explanations, then every $P(t)$ for any ground term t is a ground minimal explanation of G. In this case, $P(x)$ is preferable to any $P(t)$ because $\exists x\,(P(x))$ is a weaker condition under which G holds.

The computation of minimal explanations can be precisely characterized by using the new characteristic clauses in the next proposition.

Proposition 3.11 Let Σ, Γ and G be the same as in Definition 3.10. The set of all minimal explanations of G from (Σ, Γ) is

$$\overline{Newcarc(\Sigma, \neg G, \mathcal{P})}\,, \text{ where } \mathcal{P} = \langle\, \overline{\Gamma}\,\rangle\,.$$

Proof: Suppose that E is an explanation of G from (Σ, Γ). By Definition 3.10, it is observed that the three conditions of the explanation E can be written as

1. E is a conjunction of literals of Γ with no literal repeated,
2. $\Sigma \cup \{\, \exists E \,\} \models G$, and
3. $\Sigma \cup \{\, \exists E \,\}$ is consistent,

where $\exists F$ is a formula obtained by adding an existential quantifier for each free variable in a formula F (the *existential closure* of F). These conditions are further equivalent to the following:

1′. $\neg E$ is a clause all literals of which belong to $\overline{\Gamma}$,
2′. $\Sigma \cup \{\neg G\} \models \neg E$, and
3′. $\Sigma \not\models \neg E$.

Therefore, it holds that

$$\neg E \in [\, Th_{\mathcal{P}}(\Sigma \cup \{\neg G\}) - Th(\Sigma)\,]\,.$$

Conversely, it can be easily shown that if for a clause F,

$$F \in [\, Th_{\mathcal{P}}(\Sigma \cup \{\neg G\}) - Th(\Sigma)\,]$$

holds, then $\neg F$ is an explanation of G from (Σ, Γ).

Now, observe that for two explanations E and E' of G, E is a subexplanation of E' if and only if $\neg E$ subsumes $\neg E'$. Thus, E is a minimal explanation of G if and only if $\neg E$ is not properly subsumed by any clause from $[\, Th_{\mathcal{P}}(\Sigma \cup \{\neg G\}) - Th(\Sigma)\,]$. By Definition 3.6, E is a minimal explanation of G from (Σ, Γ) if and only if $\neg E \in Newcarc(\Sigma, \neg G, \mathcal{P})$. □

Note that in implementing resolution-based abductive procedures based on Proposition 3.11 within first-order logic, both the query G and its explanation E should be considered as existentially quantified formulas. If G contains universally quantified variables, then each of them is replaced with a new constant or function in $\neg G$ through Skolemization. Then, to get a universally quantified explanation in negating each new characteristic clause containing Skolem functions, we need to apply the *reverse Skolemization* algorithm [Cox and Pietrzykowski, 1986].

Although the production field in Proposition 3.11 is set to be $\mathcal{P} = \langle\, \overline{\Gamma}\,\rangle$ for the abductive frame work (Σ, Γ), any stable production field with a certain condition can be utilized to restrict explanations to those formulas satisfying the condition. In particular, conditions on a maximum length of clauses and on a maximum number of occurrences of some function symbols are useful when there is an infinite number of minimal explanations of a query (such as "Under what conditions does $Ancestor(x, y)$ hold?" in [Demolombe and Fariñas del Cerro, 1991, Example 4]). For example, let

$$\begin{aligned} \Sigma &= \{\ \neg P(x,y) \vee \neg P(y,z) \vee P(x,z)\ \}\,, \\ \Gamma &= \{\, P\,\}^{+}\,, \\ G &= P(A,B)\,. \end{aligned}$$

There is an infinite number of minimal explanations of G from (Σ, Γ):

$$P(A,B), \quad P(A,x) \wedge P(x,B), \quad P(A,x) \wedge P(x,y) \wedge P(y,B), \quad \ldots\,.$$

If the production field is given as

$$\mathcal{P} = \langle\, \{P\}^{-},\ \textit{length is fewer than } 3\,\rangle\,,$$

then $Newcarc(\Sigma, \neg G, \mathcal{P})$ contains only two clauses:

$$\neg P(A,B), \quad \neg P(A,x) \vee \neg P(x,B)\,.$$

Proposition 3.12 Let Σ, Γ and G be the same as in Definition 3.10. The following three statements are equivalent.
(1) There is an explanation of G from (Σ, Γ).
(2) There is a ground explanation of G from (Σ, Γ).
(3) There is a minimal explanation of G from (Σ, Γ).

Proof: (1) $\Leftrightarrow$ (2). We prove (1) $\Rightarrow$ (2) because the opposite is obvious. Let E be an explanation of G. We can assume that E contains free variables; otherwise the proof is completed. Since $\Sigma \cup \{E\}$ is consistent, there is a model M of Σ such that M satisfies a ground instance $E\theta$ of E for some substitution θ. Therefore, $\Sigma \cup \{E\theta\}$ is satisfiable. Since $\Sigma \cup \{E\} \models G$, it holds that $\Sigma \cup \{E\theta\} \models G$. Hence, $E\theta$ is a ground explanation of G.

(1) $\Leftrightarrow$ (3). We prove (1) $\Rightarrow$ (3) because the opposite is obvious. Let E be an explanation of G and $\mathcal{P} = \langle\, \overline{\Gamma}\, \rangle$. Notice that $\mathcal{P}$ is a stable production field. By Proposition 3.11, $\neg E$ is contained in $[\, Th_{\mathcal{P}}(\Sigma \cup \{\neg G\}) - Th(\Sigma)\,]$. By Proposition 3.5, there is a clause F in $Carc(\Sigma \cup \{\neg G\}, \mathcal{P})$ such that F subsumes $\neg E$. Then, $E' = \neg F$ is a conjunction of literals of Γ such that $\Sigma \cup \{E'\} \models G$. Since $\Sigma \cup \{E\}$ is consistent and an instance of E' is contained in E, it holds that $\Sigma \cup \{E'\}$ is consistent. Therefore, E' is an explanation of G and is a subexplanation of E because $\neg E'$ subsumes $\neg E$. Since $\neg E' = F$ is not properly subsumed by any clause from $Th_{\mathcal{P}}(\Sigma \cup \{\neg G\})$, E' is a minimal explanation of G. $\Box$

The notion of extension of an abductive framework is useful for a kind of *default reasoning* if each hypothesis can be regarded as a default [Poole, 1988]. The computation of membership in some (or no) extension can be characterized again by using the new characteristic clauses in the next proposition.

Proposition 3.13 Let Σ, Γ and G be the same as in Definition 3.10. There is an extension of (Σ, Γ) in which G holds if and only if

$$Newcarc(\Sigma, \neg G, \mathcal{P}) \neq \emptyset\,, \text{ where } \mathcal{P} = \langle\, \overline{\Gamma}\, \rangle\,.$$

Proof: By the definition, for each extension $\mathbf{X}$ of (Σ, Γ), there is a maximal set $\mathbf{M}$ of ground literals of Γ such that $\Sigma \cup \mathbf{M}$ is consistent. By the compactness theorem of first-order logic (see [Enderton, 1972, Section 2.5]), $G \in \mathbf{X}$ if and only if there is a finite conjunction $E_{\mathbf{M}'}$ of elements of $\mathbf{M}$ such that $E_{\mathbf{M}'}$ is an explanation of G [Poole, 1988, Theorem 2.6]. Therefore, G has a ground explanation from (Σ, Γ) if and only if G holds

in some extension of (Σ, Γ). By Proposition 3.12, there is no extension of (Σ, Γ) in which G holds if and only if there is no minimal explanation of G from (Σ, Γ), i.e.,

$$Newcarc(\Sigma, \neg G, \mathcal{P}) = \emptyset$$

by Proposition 3.11. Hence, the proposition holds. □

3.3.2 Prime Implicates

This subsection presents the basic notion of consequence-finding within the propositional calculus. In the propositional case, the set of atomic formulas $\mathcal{A}$ is reduced to the set of propositional symbols in the language. Here we assume that $\mathcal{A}$ is finite. The subsumption criterion is now very simple: a clause C *subsumes* a clause D if $C \subseteq D$. In this case, C subsumes D if and only if $\models C \supset D$.

Definition 3.14 Let Σ be a set of propositional clauses.
(1) A clause C is an *implicate* of Σ if it is a theorem of Σ (i.e., $\Sigma \models C$).
(2) $Th(\Sigma)$ is the set of implicates of Σ.
(3) The *prime implicates* of Σ are: $PI(\Sigma) = \mu\, Th(\Sigma)$.

The *characteristic clauses* of Σ with respect to $\mathcal{P}$ are all the unsubsumed implicates of Σ that belong to $\mathcal{P}$. In other words, $Carc(\Sigma, \mathcal{P})$ is the set of prime implicates of Σ belonging to $\mathcal{P}$. When $\mathcal{P} = \mathcal{P}_{\mathcal{L}} = \langle\, \mathcal{L}\, \rangle$, it holds that

$$Carc(\Sigma, \mathcal{P}_{\mathcal{L}}) = PI(\Sigma)\,.$$

On the contrary, Proposition 3.11 shows that $Newcarc(\Sigma, F, \mathcal{P})$ represents *abduction*, that is, the set of minimal explanations of $\neg F$ from $(\Sigma, \overline{\mathbf{L}})$, where $\mathbf{L}$ is the characteristic literals of $\mathcal{P}$.

The next lemma says that the set Σ of clauses is logically equivalent to $PI(\Sigma)$ in the sense that both sets can produce the same (new) characteristic clauses with respect to a production field $\mathcal{P}$.

Lemma 3.15 For any stable production field $\mathcal{P}$,

$$Newcarc(\Sigma, C, \mathcal{P}) = Newcarc(PI(\Sigma), C, \mathcal{P})\,.$$

Proof: Firstly,

$$\begin{aligned} Carc(PI(\Sigma), \mathcal{P}) &= Carc(Carc(\Sigma, \mathcal{P}_{\mathcal{L}}), \langle\, \mathbf{L}, Cond\, \rangle) \\ &= Carc(\Sigma, \langle\, \mathcal{L} \cap \mathbf{L}, Cond\, \rangle) \\ &= Carc(\Sigma, \mathcal{P})\,. \end{aligned}$$

Now,

$$\begin{aligned} Carc(PI(\Sigma) \cup \{C\}, \mathcal{P}) &= \mu Th_{\mathcal{P}}(\mu Th(\Sigma) \cup \{C\}) \\ &= \mu Th_{\mathcal{P}}(Th(\Sigma) \cup \{C\}) \\ &= Carc(\Sigma \cup \{C\}, \mathcal{P}) . \end{aligned}$$

The lemma follows immediately by Proposition 3.7. □

Note that the *prime implicants* of a disjunctive normal form formula can be defined in the same manner if the duality of $\wedge$ and $\vee$ is taken into account. The notion of prime implicants/implicates was originally investigated for uses in minimizing Boolean functions of switching circuits [Quine, 1952]. Additional background can be found in [Tison, 1967; Kean and Tsiknis, 1990]. Computing prime implicates is also an essential task in the ATMS [de Kleer, 1986] and in its generalization, the *clause management system* (CMS) [Reiter and de Kleer, 1987], which will be investigated in the next subsection.

3.3.3 CMS/ATMS

An *assumption-based truth maintenance system* (ATMS) [de Kleer, 1986] has been widely used when problems require reasoning in multiple contexts. However, this basic ATMS can handle only the restricted form of formulas, and is described algorithmically rather than declaratively, and no proof of its correctness is given, so it is not obvious how to generalize or refine it. Thus it is desirable to formalize generalizations of the ATMS within simple model and proof theories.

Recent investigations shows that there are strong connections between an ATMS and a logical account of *abduction*. Reiter and de Kleer [1987] pointed out that the task of the CMS/ATMS can be viewed as abduction in the propositional case. Abductive characterizations of ATMSs can also be seen in [Inoue, 1989; Levesque, 1989; Selman and Levesque, 1990; Inoue, 1990b]. In [Inoue, 1989], an ATMS is characterized by an abductive framework (Σ, Γ) of Definition 3.10: it is precisely intended to generate *all and only* minimal explanations from (Σ, H). In the ATMS terminology, the set of minimal explanations of a *node* G from the *justifications* Σ and the *assumptions* Γ is called the *label* of G, which is *consistent, sound, complete* and *minimal*. The *basic* ATMS [de Kleer, 1986] is restricted to accepting only Horn clause justifications and atomic assumptions. In the declarative conditions for an ATMS (to be given in Definition 3.23), justifications can contain non-Horn clauses, and assumptions are allowed to be literals, so that this generalization covers de Kleer's [1988; 1989] various extended versions of the ATMS, Dressler's [1989] extended basic ATMS (with nogood inference), and Reiter and de Kleer's [1987] *clause management system* (CMS).

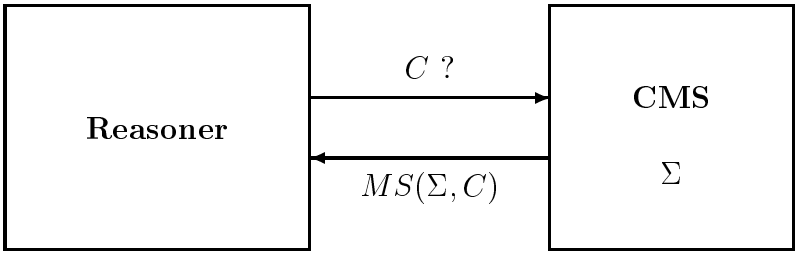

Figure 3.1: A Reasoner-CMS architecture

CMS

Now, let us first look at a CMS defined by Reiter and de Kleer [1987]. A CMS is intended to work together with a reasoner, which issues queries that take the form of clauses. The CMS is then responsible for finding *minimal supports* for the queries:

Definition 3.16 [Reiter and de Kleer, 1987] Let Σ be a set of clauses and C a clause. A clause S is a *support for* C *with respect to* Σ if

1. $\Sigma \models S \cup C$, and
2. $\Sigma \not\models S$.

A support S for C with respect to Σ is *minimal* if no other support S' for C subsumes S. The set of minimal supports for C with respect to Σ is denoted as $MS(\Sigma, C)$.

Figure 3.1 illustrates an architecture for a reasoning system coupled with a CMS.

Comparing minimal supports with minimal explanations in Definition 3.10, a minimal support for C with respect to Σ is exactly the negation of a minimal explanation of C from the abductive framework $(\Sigma, \mathcal{L})$. That is,

Proposition 3.17 Let Σ be a set of clauses and C a clause.

$$MS(\Sigma, C) = Newcarc(\Sigma, \neg C, \mathcal{P}_{\mathcal{L}}) .$$

Proof: A clause S is a support for C with respect to Σ

$\Leftrightarrow$ $\Sigma \models S \cup C$, and $\Sigma \not\models S$

$\Leftrightarrow$ $\Sigma \cup \{\neg C\} \models S$, and $\Sigma \not\models S$

$\Leftrightarrow$ $S \in [\, Th(\Sigma \cup \{\neg C\}) - Th(\Sigma)\,]$.

Therefore, $S \in MS(\Sigma, C) \Leftrightarrow S \in Newcarc(\Sigma, \neg C, \mathcal{P}_{\mathcal{L}})$ (by Definition 3.6). $\square$

Example 3.18 Suppose that Σ contains the following three clauses:

$$\neg P \vee \neg Q \vee \neg R \vee G\,,$$
$$P \vee \neg Q \vee G\,,$$
$$Q \vee \neg R \vee G\,.$$

Then, $MS(\Sigma, G)$ can be given by the following three clauses:

$$Newcarc(\Sigma, \neg G, \mathcal{P}_{\mathcal{L}}) = \{\ \neg G,\quad P \vee \neg Q,\quad \neg R\ \}\,,$$

which correspond to the minimal explanations of G: $\{\, G,\ \neg P \wedge Q,\ R\,\}$.

Using the above formulation, Definition 3.16 can be easily extended to handle any formula instead of a clause as a query [Inoue, 1990b]. Setting the production field to $\mathcal{P}_{\mathcal{L}} = \langle\, \mathcal{L}\,\rangle$, the minimal supports for a formula F can be defined as:

$$MS(\Sigma, F) = Newcarc(\Sigma, \neg F, \mathcal{P}_{\mathcal{L}})\,.$$

Equivalently, by using Proposition 3.7, the above can be written as:

$$MS(\Sigma, F) = PI(\Sigma \cup \{\neg F\}) - PI(\Sigma)\,.$$

This formulation can remove one of the limitations of the CMS. In [Reiter and de Kleer, 1987], the CMS is defined to handle only the queries of the clause form, so that it cannot deal with minimal explanations of a conjunctive query. For instance, the minimal supports for the conjunctive query $P \wedge Q$ with respect to Σ in Example 3.18,

$$\mu\,\{\,\neg E \mid \Sigma \models (E \supset (P \wedge Q)) \text{ and } \Sigma \not\models \neg E\}\,,$$

can be obtained straightforwardly in our formulation as:

$$Newcarc(\Sigma, \neg P \vee \neg Q, \mathcal{P}_{\mathcal{L}}) = \{\ \neg P \vee \neg Q,\quad \neg Q \vee G\ \}\,.$$

Also, for a disjunctive normal form query F, $MS(\Sigma, \neg F)$ can be defined using Proposition 3.8.

We thus see that the operation defined as $Newcarc$ can compute minimal supports. Note that Reiter and de Kleer [1987] consider the two ways the CMS manages the knowledge base:

1. keeping the set of clauses Σ transmitted by the reasoner as it is (the *interpreted approach*), or
2. computing $PI(\Sigma)$ (the *compiled approach*).

In the interpreted approach, we can choose the primitive $Newcarc$ operation as a basic computational task, as it is not necessary to compute either $PI(\Sigma)$ or $PI(\Sigma \cup \{\neg F\})$ to obtain $MS(\Sigma, \neg F)$ by Proposition 3.17.

On the other hand, when we are faced with a situation where we want to know explanations for many different queries, we must run the algorithm each time a query is issued. Instead of keeping the initial theory Σ as it is and doing the same deductions over and over for different queries, some of these inferences need only be made once. That is the motivation for the compiled approach: the set Σ is compiled into the *saturated* set, $PI(\Sigma) = Carc(\Sigma, \mathcal{P}_{\mathcal{L}})$. Given $PI(\Sigma)$, it is easy to find $MS(\Sigma, C)$ for each clause C in the compiled approach, without computing another saturated set $PI(\Sigma \cup \{\neg C\})$, as the following relationships have been shown between the prime implicates $PI(\Sigma)$ and minimal supports with respect to Σ:

Proposition 3.19 [Reiter and de Kleer, 1987, Theorem 5] Let C be a clause.

$$MS(\Sigma, C) = \mu \{ P - C \mid P \in PI(\Sigma) \text{ and } P \cap C \neq \emptyset \} .$$

Corollary 3.20 [Reiter and de Kleer, 1987, Corollary 4] Let U be a unit clause.

$$MS(\Sigma, U) = \{ P - U \mid P \in PI(\Sigma) \text{ and } U \subseteq P \} .$$

One of the disadvantages of the compiled approach is the high cost of updating the knowledge base. When the reasoner adds a clause C to Σ, we must compute all the $PI(\Sigma \cup \{C\})$. Thus there is a trade-off between the compiled and the interpreted approaches. In the former, the computation of $PI(\Sigma)$ is very expensive, but retrieval is then efficient. In the latter, the CMS's database is kept as it is; the price we have to pay is a high retrieval cost. Note that either computing supports with an uncompiled theory or compiling a theory is an enumeration problem of prime implicates and each computational complexity is exponential [Kean and Tsiknis, 1990; Provan, 1990].

There is another interesting production field that offers an intermediate alternative to the compiled/interpreted disjunctive. The original theory Σ can yet be simplified, computing its reduction by replacing it with the set of *sub-clauses* implied by Σ. Here, a clause C is a *sub-clause* of a clause D if $C \subseteq D$.

Definition 3.21 The *reduction* of a set of clauses Σ is:

$$Reduced(\Sigma) = Carc(\Sigma, \mathcal{P}_R) ,$$

where $\mathcal{P}_R = \langle \mathcal{L},$ *sub-clause of a clause of* $\Sigma \rangle$.

Proposition 3.22 For any stable production field $\mathcal{P}$,

$$Newcarc(\Sigma, C, \mathcal{P}) = Newcarc(Reduced(\Sigma), C, \mathcal{P}) .$$

Proof: It is easy to show that $Carc(Reduced(\Sigma), \mathcal{P}) = Carc(\Sigma, \mathcal{P})$. The proposition can be proved in the similar manner to Lemma 3.15. □

Note that $\mathcal{P}_R$ is a stable production field. The reduction of the clauses is an empirically efficient way to manage the knowledge base. The original set Σ can be reduced to a great extent; computation from $Reduced(\Sigma)$ will be much more efficient than from Σ. For example, the set $\{ P, \neg P \vee Q \}$ will be reduced to $\{ P, Q \}$ (in this case producing the sub-clauses is equivalent to saturation). However, $\{ P \vee R, \neg P \vee Q \}$ contains all the sub-clauses implied by it and thus cannot be simplified, while the saturation would have added $Q \vee R$.

ATMS

In versions of ATMSs proposed by de Kleer [1986; 1988; 1989], there is a distinguished set of *assumptions* $\Gamma \subseteq \mathcal{L}$. An ATMS can be defined as a system responsible for finding all the negations of minimal supports for the queries such that each support consists of only literals from $\overline{\Gamma}$ [Reiter and de Kleer, 1987; Inoue, 1990b]:

Definition 3.23 An *ATMS* is a triple $\langle N, \Gamma, \Sigma \rangle$, where $N \subseteq \mathcal{L}$ is a set of literals, *nodes*; $\Gamma \subseteq N$ is a set of literals, *assumptions*; and Σ is a set of clauses all of whose literals belong to $N \cup \overline{N}$, *justifications*. The *label of* $n \in N$ *with respect to* $\langle N, \Gamma, \Sigma \rangle$ is defined as:

$$Label(n, \Gamma, \Sigma) = \overline{Newcarc(\Sigma, \neg n, \mathcal{P})}, \quad \text{where } \mathcal{P} = \langle \overline{\Gamma} \rangle.$$

The following properties [de Kleer, 1986; de Kleer, 1988] hold for the label of each node $n \in N$ with respect to an ATMS $\langle N, \Gamma, \Sigma \rangle$ given by Definition 3.23:

Proposition 3.24 Let $\langle N, \Gamma, \Sigma \rangle$ be an ATMS, $n \in N$ a literal, and $\mathcal{P} = \langle \overline{\Gamma} \rangle$.
(1) *Label consistency*: For each $E_i \in Label(n, \Gamma, \Sigma)$, $\Sigma \cup \{E_i\}$ is satisfiable.
(2) *Label soundness*: For each $E_i \in Label(n, \Gamma, \Sigma)$, $\Sigma \cup \{E_i\} \models n$.
(3) *Label completeness*: For every conjunction E of assumptions from Γ, if $\Sigma \cup \{E\} \models n$, then there exists $E_i \in Label(n, \Gamma, \Sigma)$ such that E_i is a sub-conjunction of E.
(4) *Label minimality*: Every $E_i \in Label(n, \Gamma, \Sigma)$ is not a super-conjunction of any other element of $Label(n, \Gamma, \Sigma)$.

Proof: By Definition 3.23 and Proposition 3.11, $E_i \in Label(n, \Gamma, \Sigma)$ is a minimal explanation of n from (Σ, Γ). Therefore, these four properties obviously hold by Definition 3.10. □

In the ATMS, a set of assumptions inconsistent with Σ is called *nogood*. Since $\Sigma \models \neg E$ holds for nogood E, the set of *minimal* nogoods is characterized as

$$Nogood(\Gamma, \Sigma) = \overline{Carc(\Sigma, \mathcal{P})}, \text{ where } \mathcal{P} = \langle \overline{\Gamma} \rangle.$$

Thus, the label consistency (Proposition 3.24 (1)) is alternatively given as follows:

> *Label consistency'*: For each $E_i \in Label(n, \Gamma, \Sigma)$, E_i is not a super-conjunction of any element of $Nogood(\Gamma, \Sigma)$. Equivalently, $\neg E_i$ is not subsumed by any clause from $\overline{Nogood(\Gamma, \Sigma)}$.

According to [Inoue, 1990a; Inoue, 1990b], various sound and complete methods for both generating and updating the labels of queries relative to an ATMS consisting of a non-Horn theory and literal assumptions will be presented in Section 4.2.4, which are based on a consequence-finding procedure shown in the next Chapter.

3.3.4 Circumscription

We have defined abduction using the new characteristic clauses in Section 3.3.1. In Proposition 3.11, we have characterized the explanations from an abductive framework in terms of the new characteristic clauses of the negation of the query. In Proposition 3.13, we have shown the equivalence between membership in some extension and the non-emptiness of the new characteristic clauses. In this subsection, we consider another important problem of default reasoning, that is, to predict formulas that hold in all extensions.

When the theory is function-free, we have seen in Section 2.3.2 that this skeptical prediction problem is essentially equivalent to *circumscription* [McCarthy, 1980; Lifschitz, 1985] under the unique-names assumption (UNA) and the domain-closure assumption (DCA) [Gelfond *et al.*, 1989; Ginsberg, 1989]. In Chapter 5, we will see that proving that a formula holds in a circumscriptive theory [Przymusinski, 1989; Ginsberg, 1989], and other proof methods for nonmonotonic reasoning formalisms (including explanation-based argument systems [Poole, 1989] and variations of the closed-world assumption [Bossu and Siegel, 1985; Minker and Rajasekar, 1990; Gelfond *et al.*, 1989]), are based on finding explanations of the query, and showing that these explanations cannot be refuted.

Proposition 3.25 [Inoue and Helft, 1990] Let $CIRC(\Sigma; \mathbf{P}; \mathbf{Z})$ be the circumscription of predicates $\mathbf{P}$ in a theory Σ with variable predicates $\mathbf{Z}$. Set the production field to be

$$\mathcal{P}_{CIRC} = \langle \mathbf{P}^+ \cup \mathbf{Q}^+ \cup \mathbf{Q}^- \rangle,$$

where $\mathbf{Q}$ is the fixed predicates.

(1) [Gelfond *et al.*, 1989] Let F be a formula not containing literals from $\mathbf{Z}$.
$CIRC(\Sigma;\mathbf{P};\mathbf{Z}) \models F$ if and only if $Newcarc(\Sigma, F, \mathcal{P}_{CIRC}) = \emptyset$. □
(2) [Gelfond *et al.*, 1989; Ginsberg, 1989] Let F be any formula.
$CIRC(\Sigma;\mathbf{P};\mathbf{Z}) \models F$ if and only if there is a conjunction G of clauses from $Newcarc(\Sigma, \neg F, \mathcal{P}_{CIRC})$ such that $Newcarc(\Sigma, \neg G, \mathcal{P}_{CIRC}) = \emptyset$.

Proof: The proof will be given with detailed analysis in Section 5.2. □

We will consider an implementation of skeptical prediction later as theorem proving for circumscription in Chapter 5. When a query in abduction or circumscription contains existentially quantified variables, it is sometimes desirable to know for what instances of these variables the query holds. This *answer extraction* problem [Helft *et al.*, 1991] is also considered in Section 5.7 of Chapter 5 where the characteristic clauses of circumscriptive theories play an important role in computation.

3.3.5 Other AI theories

Since we have characterized the prime implicates, the CMS/ATMS, abduction and circumscription, any application area of these techniques can be directly characterized by using the notion of the (new) characteristic clauses: for instance, constraint satisfaction problems [de Kleer, 1989], principles of diagnosis [de Kleer *et al.*, 1990], synthesis [Finger, 1987; Helft and Konolige, 1990] (plan recognition, prediction, design), natural language understanding [Hobbs *et al.*, 1988; Stickel, 1989], finding first-order proofs for creating dependency networks [Junker and Konolige, 1990], and generation of conditional answers in deductive databases [Demolombe and Fariñas del Cerro, 1991]. Also, some advanced inference mechanisms such as inductive and analogical reasoning may also take abductive forms of representation [O'Rorke, 1990; Arima, 1992].

Example 3.26 Plan recognition (or plan synthesis) is an example which can be formalized by abduction (finding explanations) and consequence-finding (finding ramifications). Here is an illustration of what plan synthesis looks like (see, for example, [Helft and Konolige, 1990]). In plan synthesis, to satisfy a goal G, an agent looks for a sequence $\mathbf{A}$ of actions that can perform this goal. This problem is in essence the same as abduction: we can compute it by negating each clause in $Newcarc(\Sigma, \neg G, \mathcal{P})$, where Σ is the background theory and $\mathcal{P}$ represents the action vocabulary. Then, the obtained plan $\mathbf{A}$ should be added to the theory to check whether an unintended effect is caused. For example, to remove $Block(A)$ from the table, $Clear(Table)$ would perform this goal, but this plan might cause unintended side effects. This ramification can be found from $Newcarc(\Sigma, \mathbf{A}, \mathcal{P}')$, where $\mathcal{P}'$ represents the event vocabulary that may be distinct from the action vocabulary $\mathcal{P}$.

Chapter 4

Linear Resolution for Consequence-finding

In this chapter, we show an extension of the Model Elimination theorem proving procedure for consequence-finding. The extension is called SOL-resolution (Skipping Ordered Linear resolution). Although linear resolution involving C-ordering (or C-ordered linear resolution) such as Model Elimination is complete for refutation, it is not complete for consequence-finding. We show how adding a skip rule to Model Elimination makes it complete for consequence-finding and for characteristic-clause-finding. An important feature of SOL-resolution is that it constructs characteristic clauses, which is a subset of the consequences, directly without testing each generated clause for the required property. Compared with set-of-support resolution, SOL-resolution generates fewer clauses to find such a subset of consequences. In the propositional case, this is an elegant tool for computing the prime implicants/implicates. This feature is very effective for computing abduction and nonmonotonic reasoning. Efficient but incomplete variations of SOL-resolution and their abilities to find the most-specific and the least-specific abductive explanations are also discussed in this chapter, where computational complexity is also taken into account.

4.1 Introduction

In Section 3.1, we have seen three major problems for consequence-finding. The last chapter is devoted to show a solution to the third problem, that is, to show the usefulness of consequence-finding, in particular, the finding of those particular theorems belonging to production fields. In this chapter, we will give solutions to the first and the second problems. Namely, we will consider the questions:

1. How can we get a C-ordered linear resolution procedure that is complete for consequence-finding?

2. How can we find (new) characteristic clauses directly without generating and testing all derivable theorems?

These two problems can be solved at once if we give an answer to the question:

> How can we compute (new) characteristic clauses efficiently by using C-ordered linear resolution?

Since most works in theorem proving have been concerned with techniques for refutation, there have been only a few work on consequence-finding such as [Lee, 1967; Slagle *et al.*, 1969; Minicozzi and Reiter, 1972]. More recently, some researchers have began to pay more attention to consequence-finding. In particular, the following work can be related to the above problem:

- Finger [1987] gives two complete procedures based on set-of-support resolution, one for generating formulas (*ramification*) derivable from a set of formulas and a newly added formula as an initial set of support (the RGC procedure), and the other for finding residues (resolution residue). However, these procedures do not utilize C-ordering. Finger also gives another procedure based on C-ordered set-of-support resolution (ordered residue), but it can be used only for Horn clauses.

- Bossu and Siegel [1985] propose a complete algorithm for finding the set of positive clauses derivable from groundable formulas by using their saturation algorithm based on A-ordering. More recently, Siegel [1987] proposed a complete algorithm for finding the characteristic clauses by adding one rule called *skip* operation to C-ordered linear resolution. This algorithm is, although propositional, much more efficient than Bossu and Siegel's saturation algorithm, and realizes direct search for characteristic clauses.

- Przymusinski [1989] defines MILO-resolution, which outputs new characteristic clauses to be used in a query answering procedure for circumscription in ground clauses. MILO-resolution can also be characterized as an instance of C-ordered linear resolution with the skip operation [Inoue and Helft, 1990].

While these previous procedures are important, none of them achieves all of our goals. Namely, there has been no C-ordered linear resolution procedure complete for finding the characteristic clauses in the general case. In this chapter, we will provide such a procedure, and thus will give satisfactory solutions to the three problems given in Section 3.1 in the last chapter:

1. We provide *SOL-resolution*, C-ordered linear resolution with the skip rule, which lifts Siegel's [1987] procedure for the general case, and its completeness result for consequence-finding. Compared with set-of-support resolution, on which Finger's [1987] procedures are based, SOL-resolution contains more restriction strategies and generates fewer clauses to find (new) characteristic clauses.

2. We show that an easy modification of SOL-resolution, where one of the operations is preferred over others so as to reduce the search space, can be shown to be applicable to broad, more efficient variations of consequence-finding.

3. We show how SOL-resolution can be well applied to generate interesting formulas for abductive and nonmonotonic reasoning. Applications of SOL-resolution to the CMS/ATMS is also demonstrated in this chapter, while theorem proving for circumscription based on SOL-resolution will be discussed in Chapter 5.

Among other possibilities, *abduction* may be one of the most important applications of SOL-resolution. Although most first-order abductive procedures [Pople, 1973; Cox and Pietrzykowski, 1986; Poole *et al.*, 1987; Stickel, 1989; Poole, 1991] are based on C-ordered linear resolution, no completeness results have been obtained for abductive explanations generated by them. More recently, Demolombe and Farinãs del Cerro [1991] independently provided a complete procedure for a special kind of abduction, where every literal can be hypothesized. However, it uses a more primitive style of resolution so that it is hard to connect it with previous abductive procedures. We can get the completeness result for an abductive procedure by characterizing abduction as characteristic-clause-finding for which SOL-resolution is complete. Abductive completeness is very important and is crucial for some applications of abduction. For example, to compute circumscription we can use an abductive procedure, but we require the procedure to generate every desired explanation.

This chapter is organized as follows. Section 4.2 presents the basic procedure based on C-ordered linear resolution which is sound and complete for characteristic-clause-finding. Efficient but incomplete variations of the basic procedure and their properties are provided in Section 4.3, where computational complexity is also taken into account. The proofs of two main theorems (Theorems 4.3 and 4.18) are deferred to Section 4.5.

4.2 Skipping Ordered-Linear Resolution

In this section, we show the basic procedure for implementing the primitive *Newcarc* operation by using an extension of *C-ordered linear resolution*. By the term C-ordered linear resolution, we mean all variants of *Model Elimination* procedures [Loveland,

1969; Loveland, 1978], that is, the family of linear resolution using ordered clauses and the information of literals resolved upon. Other examples of C-ordered linear resolution are SL-resolution [Kowalski and Kuhner, 1971], OL-resolution [Chang and Lee, 1973], the GC procedure [Shostak, 1976], and SLI-resolution [Minker and Rajasekar, 1990]. This family is recognized to be one of the most efficient classes of resolution for non-Horn theories because it contains several restriction strategies, and can be viewed as a predecessor of Prolog's SLD(NF)-resolution [Lloyd, 1984]. It has become important to theorem proving partly because the Prolog implementation architecture applies to these procedures, giving them very high inference rates. For example, PTTP [Stickel, 1988] is a notable Model Elimination theorem prover and is extremely fast.

An important feature of the procedure we present in Section 4.2.1 is that it is *direct*, namely it is both sensitive to the given clause added to the theory and restricted to searching only characteristic clauses. While both $Newcarc(\Sigma, F, \mathcal{P})$ for a CNF-formula F and $Carc(\Sigma, \mathcal{P})$ can be computed by using primitive $Newcarc$ operations as seen in Propositions 3.8 and 3.9, we will show in Sections 4.2.2 and 4.2.3 that these can also be computed directly by using the basic procedure.

4.2.1 SOL-Resolution

Given a set of clauses Σ, a stable production field $\mathcal{P}$ and a clause C, we show a procedure to compute $Newcarc(\Sigma, C, \mathcal{P})$, which is an extension of C-ordered linear resolution. There are two reasons why C-ordered linear resolution is, among other strategies, useful for computing the new characteristic clauses:

1. A newly added single clause C can be treated as the *top clause* of a linear deduction. This is a desirable feature for consequence-finding since the procedure can directly derive the theorems relevant to the added information.

2. It is easy to achieve the requirement that the procedure should focus on producing only those theorems that belong to $\mathcal{P}$. This is implemented by allowing the procedure to *skip* the selected literals belonging to $\mathcal{P}$ in a linear deduction. Compared to resolution residue [Finger, 1987] which is based on set-of-support resolution, the computational superiority of our technique comes from this property of directing search to $\mathcal{P}$, in addition to the fact that C-ordered linear resolution contains more natural restriction strategies.

Some procedures are known to perform the above computation for restricted theories. For *propositional* theories, Siegel [1987] shows a complete algorithm by adding the skip operation to SL-resolution. For *ground* theories with a *particular* production

field for circumscription (described in Proposition 3.25), Inoue and Helft [1990] point out that Przymusinski's [1989] MILO-resolution, an extension of Chang and Lee's OL-resolution, can be viewed as C-ordered linear resolution with the skip operation. The same argument can be applied to Minker and Rajasekar's [1990] SLINF-resolution, a modified version of SLI-resolution.

The following inference system called *SOL (Skipping Ordered Linear) resolution* is a kind of generalization of [Przymusinski, 1989; Siegel, 1987; Minker and Rajasekar, 1990], again in the sense that produced clauses are not limited to some particular applications and that it deals with the general case. The description below is mainly based on terminology of OL-resolution [Chang and Lee, 1973], but the result is not restricted to its extension. [1] An *ordered* clause is a sequence of literals possibly containing *framed literals* which represent literals that have been resolved upon. From a clause C an ordered clause $\vec{C}$ is obtained just by ordering the elements of C; conversely, from an ordered clause $\vec{C}$ a clause C is obtained by removing the framed literals and converting the remainder (i.e., the sequence of the unframed literals) into the set (i.e., disjunction) of the literals. We assume throughout this chapter that literals are ordered from left to right in each ordered clause. A *structured* clause $\langle\, P,\ \vec{Q}\,\rangle$ is a pair of a clause P and an ordered clause $\vec{Q}$, whose clausal meaning is $P \cup Q$.

Definition 4.1 (SOL-Deduction) Given a set of clauses Σ, a clause C, and a production field $\mathcal{P}$, an *SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$* consists of a sequence of structured clauses, $D_0, D_1, \ldots, D_n$, such that:

1. $D_0 = \langle\, \Box,\ \vec{C}\,\rangle$.
2. $D_n = \langle\, S,\ \Box\,\rangle$.
3. For each $D_i = \langle\, P_i,\ \vec{Q_i}\,\rangle$, $P_i \cup Q_i$ is not a tautology.
4. For each $D_i = \langle\, P_i,\ \vec{Q_i}\,\rangle$, Q_i is not subsumed by any Q_j with the empty substitution, where $D_j = \langle\, P_j,\ \vec{Q_j}\,\rangle$ is a previous structured clause, $j < i$.
5. For each $D_i = \langle\, P_i,\ \vec{Q_i}\,\rangle$, P_i belongs to $\mathcal{P}$.
6. $D_{i+1} = \langle\, P_{i+1},\ \vec{Q_{i+1}}\,\rangle$ is generated from $D_i = \langle\, P_i,\ \vec{Q_i}\,\rangle$ according to the following steps:

[1] In the terminology of Model Elimination, ordered clause, framed literal and unframed literal are called *chain*, *A-literal* and *B-literal*. The description of OL-resolution in [Chang and Lee, 1973] contains an error for the nonground case discussed later in Remark (4).

(a) Let l be the *left-most* literal of $\vec{Q}_i$. P_{i+1} and $\vec{R}_{i+1}$ are obtained by applying one of the rules:

i. **(Skip)** If $P_i \cup \{l\}$ belongs to $\mathcal{P}$, then $P_{i+1} = P_i \cup \{l\}$ and $\vec{R}_{i+1}$ is the ordered clause obtained by removing l from $\vec{Q}_i$.

ii. **(Resolve)** If there is a clause B_i in $\Sigma \cup \{C\}$ such that $\neg k \in B_i$ and l and k are unifiable with mgu θ, then $P_{i+1} = P_i\theta$ and $\vec{R}_{i+1}$ is an ordered clause obtained by concatenating $\vec{B}_i\theta$ and $\vec{Q}_i\theta$, framing $l\theta$, and removing $\neg k\theta$ (*extension*).

iii. **(Reduce)** If either

A. P_i or $\vec{Q}_i$ contains an unframed literal k that is either different from l (*factoring*) or another occurrence of l (*merge*), or

B. $\vec{Q}_i$ contains a framed literal $\boxed{\neg k}$ (*ancestry*),

and l and k are unifiable with mgu θ, then $P_{i+1} = P_i\theta$ and $\vec{R}_{i+1}$ is obtained from $\vec{Q}_i\theta$ by deleting $l\theta$.

(b) $\vec{Q}_{i+1}$ is obtained from $\vec{R}_{i+1}$ by deleting every framed literal not preceded by an unframed literal in the remainder (*truncation*).

Example 4.2 Let us consider the following theory Σ:

$$\begin{aligned} &(1) \quad \neg P \vee \neg Q \vee G\,, \\ &(2) \quad \neg U \vee \neg Q \vee P\,, \\ &(3) \quad \neg V \vee Q \vee G\,, \\ &(4) \quad R \vee \neg V \vee W\,, \end{aligned}$$

and the production field $\mathcal{P}$:

$$\mathcal{P} = \langle\, \{\, \neg U,\, \neg V,\, \neg W \,\} \,\rangle\,.$$

The following is an SOL-deduction of the clause $\neg U \vee \neg V$ from $\Sigma + \neg G$ and $\mathcal{P}$. Here, each underlined literal denotes a selected literal in the next step.

(5)	$\langle\, \Box\,,$	$\underline{\neg G}\,\rangle\,,$	top clause
(6)	$\langle\, \Box\,,$	$\underline{\neg P} \vee \neg Q \vee \boxed{\neg G}\,\rangle\,,$	resolution with (1)
(7)	$\langle\, \Box\,,$	$\underline{\neg U} \vee \neg Q \vee \boxed{\neg P} \vee \neg Q \vee \boxed{\neg G}\,\rangle\,,$	resolution with (2)
(8)	$\langle\, \neg U\,,$	$\underline{\neg Q} \vee \boxed{\neg P} \vee \neg Q \vee \boxed{\neg G}\,\rangle\,,$	skip
(9a)	$\langle\, \neg U\,,$	$\boxed{\neg P} \vee \neg Q \vee \boxed{\neg G}\,\rangle\,,$	merge
(9b)	$\langle\, \neg U\,,$	$\underline{\neg Q} \vee \boxed{\neg G}\,\rangle\,,$	truncation
(10)	$\langle\, \neg U\,,$	$\underline{\neg V} \vee G \vee \boxed{\neg Q} \vee \boxed{\neg G}\,\rangle\,,$	resolution with (3)
(11)	$\langle\, \neg U \vee \neg V\,,$	$\underline{G} \vee \boxed{\neg Q} \vee \boxed{\neg G}\,\rangle\,,$	skip
(12a)	$\langle\, \neg U \vee \neg V\,,$	$\boxed{\neg Q} \vee \boxed{\neg G}\,\rangle\,,$	ancestry
(12b)	$\langle\, \neg U \vee \neg V\,,$	$\Box\,\rangle\,.$	truncation

Remarks. (1) At Rule 6a, instead of selecting the left-most literal, we can choose the selected literal l with more liberty by employing *selected functions*, like SL-resolution [Kowalski and Kuhner, 1971].

(2) Rule 4 is included for efficiency. It does not affect the completeness described below. Notice that when this rule is applied the substitution must be empty. This strategy of deleting subsumed clauses can be partially implemented with the following restriction rule: when $\vec{Q}_i$ can be resolved against an ordered clause $\vec{B}_i$ from Σ with mgu θ and if $B_i\theta$ contains a literal $k\theta$ that appears in $\vec{Q_i}\theta$ as the framed literal $\boxed{k\theta}$ at the right of the literal resolved upon, then this resolution can be avoided, as **Resolve** in such a case would result in a clause subsumed by some previous clause in the deduction. The corresponding deletion rule for proof-finding is overlooked in the definition of OL-deduction [Chang and Lee, 1973] (and in MILO-resolution [Przymusinski, 1989]), but is clearly present in Model Elimination [Loveland, 1978] (and in Siegel's [1987] algorithm). Note that, along the lines of the Model Elimination procedure, Rules 3 and 4 may be replaced with pruning criteria (the *acceptability* condition[2]) described in terms of framed and unframed literals in an ordered clause.

(3) When the given production field $\mathcal{P}$ is in the form of $\langle\, \mathbf{L} \,\rangle$, factoring (6(a)iiiA) can be omitted in intermediate deduction steps, like Weak Model Elimination [Loveland, 1978]. Only at the final step, factoring may be performed for P_i in a structured clause of the form $\langle\, P_i,\ \Box \,\rangle$. In this case, Rules 3 and 4 are either omitted or replaced with pruning criteria (the *admissibility* condition) used in Weak Model Elimination.

(4) At Rule 6a, exactly one of the rules, Rules 6(a)i, 6(a)ii and 6(a)iii, is selected, but the three rules must be alternatives; for l belonging to $\mathcal{P}$, any rule may be applied; for l not belonging to $\mathcal{P}$, both **Resolve** and **Reduce** may be applicable. This is not a straightforward generalization of MILO-resolution or Siegel's algorithm, because they do not deal with **Reduce** as an alternative to the other two rules, but make the ancestry and merge operations mandatory after **Skip** or **Resolve** is applied. [3] Both Przymusinski and Siegel claim that the lifting lemma should work for their procedures. Unfortunately, their claims are wrong: this simpler treatment violates the completeness as described below. [4] Furthermore, even if we don't consider consequence-finding, OL-resolution [Chang and Lee, 1973], which also handles the ancestry rule as a subsequent rule of **Resolve**, is incomplete for proof-finding. For

[2] By using this condition, we can prune every ordered clause containing two literals, either of the same (framed-framed or unframed-unframed) or different (framed-unframed or unframed-framed) types, that are either identical or complementary. We need a minor change of the condition to take skipped literals into account for Rule 3.

[3] Furthermore, MILO-resolution prefers **Skip** to **Resolve**. See also Section 4.3.2.

[4] The incompleteness of MILO-resolution for first-order theories leads to the unsoundness as well as the incompleteness of the query-answering algorithm for circumscription. This observation has been reported in [Iwanuma *et al.*, 1989].

example, when the theory Σ is given as:

$$\begin{array}{ll} (1) & P(A) \vee P(x) \vee \neg Q(x)\,, \\ (2) & \neg P(B)\,, \\ (3) & Q(B)\,, \end{array}$$

it is easy to see that $\Sigma \models P(A)$. However, there is no OL-refutation from $\Sigma + \neg P(A)$. An OL-deduction from $\Sigma + \neg P(A)$ is as follows:

$$\begin{array}{lll} (4) & \underline{\neg P(A)}\,, & \text{given top clause} \\ (5) & \underline{P(x)} \vee \neg Q(x) \vee \boxed{\neg P(A)}\,, & \text{resolution with (1)} \\ (6) & \underline{\neg Q(A)} \vee \boxed{\neg P(A)}\,. & \text{ancestry} \end{array}$$

Another OL-deduction resolves $\neg P(A)$ of (4) against the second literal $P(x)$ of the clause (1). In both cases, the clause (6) is the dead-end of OL-deductions. Hence, unless the substitution θ is empty, i.e., the literal l and the ancestor goal $\neg k$ are exactly complementary, the reduction rule must be an alternative to the other rules. Model Elimination and SL-resolution deal with the reduction rule as an alternative.

(5) Rule 5 is necessary for the following reason. Skipped literals are inherited from one step to the next in the case of **Resolve** or **Reduce** rules (by $P_{i+1} = P_i\theta$). If the condition *Cond* in $\mathcal{P} = \langle\, \mathbf{L}, \mathit{Cond} \,\rangle$ is given as *"has fewer than k occurrences of the constant A"* (note that this production field is stable), then P_i may satisfy *Cond*, but $P_{i+1} = P_i\theta$ may not. Note that if *Cond* is either empty or specified in terms of a maximum length of clauses (like Example 3.2 (3)), then only the condition that $P_i \cup \{l\}$ belongs to $\mathcal{P}$ in the **Skip** rule is necessary to ensure that P_{i+1} always belongs to $\mathcal{P}$ so that Rule 5 can be omitted.

The **Skip** rule (6(a)i) reflects the following operational interpretation of a *stable* production field $\mathcal{P}$. By Definition 3.1 (5), assuming that a clause C subsumes a clause D, if D belongs to $\mathcal{P}$, then C also belongs to $\mathcal{P}$. In other words, if C does not belong to $\mathcal{P}$, then D does not belong to $\mathcal{P}$ either. Now, consider first a ground SOL-deduction, $D_0, \ldots, D_n$. For each $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$ $(0 \leq i \leq n-1)$ and for any $D_j = \langle\, P_j,\ \vec{Q}_j \,\rangle$ $(i < j \leq n)$, we see that P_i subsumes P_j with the empty substitution (i.e., $P_i \subseteq P_j$). This implies that if P_i does not belong to $\mathcal{P}$, neither does P_j [Siegel, 1987]. That is why Rule 5 and the condition in the **Skip** rule are important. The simple condition in **Skip** requiring $P_i \cup \{l\}$ to belong to $\mathcal{P}$ contributes to reducing the search space as follows. If no rule can be applied to the selected literal l of a structured clause D_i, the branch is immediately pruned. If **Skip** was applied anyway to continue the deduction, any resultant sequence would not succeed, thus making unnecessary computation.

A stable production field with a condition on a maximum length of clauses, $\mathcal{P} = \langle \mathbf{L},\ \textit{length is fewer than } k \rangle$, is practically very useful for minimizing the search effort, because it causes earlier pruning in SOL-deduction sequences. In a general SOL-deduction, however, a result of an application of **Resolve** or **Reduce** to some $D_i = \langle P_i,\ \vec{Q_i} \rangle$ with mgu θ may make the length of $P_{i+1} = P_i\theta$ shorter than the length of P_i so that P_i does not subsume P_{i+1}. This means that even if P_i does not belong to $\mathcal{P}$, P_{i+1} may belong to $\mathcal{P}$. Fortunately, we can still prune P_i in such a case by ignoring the possibility of a deduction of a shorter clause P_j $(j > i)$, as it is ensured that there exists an SOL-deduction reaching P_j in which factoring is applied before P_i is created. In fact, we can prove the completeness of SOL-resolution described below by considering only those SOL-deductions in which every P_i has no more literals than P_j $(j > i)$ (see the proof of Theorem 4.3 (2) in Section 4.5).

In addition to the above advantage of using the information on the production field $\mathcal{P}$ against the selected literal at Rule 6a, C-ordering itself is very important for reducing the search space. [5] It has been recognized that C-ordering is very restrictive in refutation. For example, $\neg L_1 \vee \cdots \vee \neg L_m$ can be refuted in only one way from m unit clauses $\{L_1, \ldots, L_m\}$ with C-ordering, but in $m!$ ways without it. In consequence-finding, for a very simple example, suppose that we are to deal with the structured clause $\langle P_i,\ \vec{Q_i} \rangle$, where $Q_i = L_1 \vee L_2 \vee \cdots \vee L_m$ and $P_i \cup \{L_1\}$ does not belong to $\mathcal{P}$. If the selected literal L_1 can be neither reduced nor resolved upon against clauses of the theory in such a way that the result of the deduction produces a clause belonging to $\mathcal{P}$, SOL-deduction will never try to examine the next literal L_2. On the other hand, the set-of-support strategy conventionally does not use C-ordering that gives priority to L_1 over L_2. It will, therefore, try all the possible operations on L_2 as well, making unnecessary computation.

We call the process of finding SOL-deductions *SOL-resolution*. The soundness and the completeness of SOL-resolution can be shown by the next theorem.

Theorem 4.3 (*Soundness and Completeness of SOL-Resolution*)
(1) (Soundness of SOL-Resolution)
If a clause S is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then S belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$.
(2) (Completeness of SOL-Resolution)
If a clause T does not belong to $Th_{\mathcal{P}}(\Sigma)$, but belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S subsumes T.

Proof: See Section 4.5 for the proof. □

[5] We should also mention that the **Skip** rule can be applied to other, superior versions of C-ordered linear resolution, such as the GC procedure [Shostak, 1976], and further improvements on these methods can be used to improve efficiency still more.

The soundness theorem guarantees that every produced clause belongs to the given production field so that we can avoid "generate-and-test" manners. Recall that C-ordered linear resolution is refutation-complete as shown, for example, in [Loveland, 1978], but is incomplete for consequence-finding [Minicozzi and Reiter, 1972]. Theorem 4.3 (2) says that SOL-resolution is complete for characteristic-clause-finding, and thus complete for consequence-finding when $\mathcal{P}$ is $\mathcal{P}_{\mathcal{L}}$ because it includes the additional skip operation.

Note that the completeness of SOL-resolution depends on whether T is a theorem of Σ or not. That is, if it holds that $\Sigma \models T$ for a clause T belonging to $\mathcal{P}$, then there is not necessarily an SOL-deduction of S subsuming T from $\Sigma + C$ and $\mathcal{P}$. This is just the same as the refutation completeness of C-ordered resolution as refutation from $\Sigma + C$ cannot be guaranteed for an unsatisfiable Σ (see Theorem 4.23).

Example 4.4 Suppose we are given the theory Σ:

$$\begin{array}{ll} (1) & \neg R \vee \neg P\,, \\ (2) & \neg R \vee \neg Q\,, \end{array}$$

and the clause C:

$$P \vee Q\,.$$

There is no OL-deduction of $\neg R$ from $\Sigma + C$, but $\neg R$ is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}_{\mathcal{L}}$ as:

(3)	$\langle\ \Box\,,\ \underline{P} \vee Q\ \rangle$,	given top clause
(4)	$\langle\ \Box\,,\ \underline{\neg R} \vee \boxed{P} \vee Q\ \rangle$,	resolution with (1)
(5a)	$\langle\ \neg R\,,\ \boxed{P} \vee Q\ \rangle$,	skip
(5b)	$\langle\ \neg R\,,\ \underline{Q}\ \rangle$,	truncation
(6)	$\langle\ \neg R\,,\ \underline{\neg R} \vee \boxed{Q}\ \rangle$,	resolution with (2)
(7a)	$\langle\ \neg R\,,\ \boxed{Q}\ \rangle$,	merge
(7b)	$\langle\ \neg R\,,\ \Box\ \rangle$.	truncation

Note that an OL-deduction would stop at (4).

4.2.2 Computing New Characteristic Clauses

We now show exactly how $Newcarc(\Sigma, F, \mathcal{P})$ for a formula F can be computed by using SOL-resolution. In contrast to incremental saturation procedures (e.g., [Bossu and Siegel, 1985]), our approach is not a naive implementation of Definition 3.6 or Proposition 3.7 that constructs both saturated sets, $Carc(\Sigma, \mathcal{P})$ and $Carc(\Sigma \cup \{C\}, \mathcal{P})$. Firstly, we define the set of clauses derivable by SOL-resolution.

Definition 4.5 Given a set of clauses Σ, a clause C, and a stable production field $\mathcal{P}$, we denote by $\Delta(\Sigma, C, \mathcal{P})$ the clauses derived by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$, that is,

$$\Delta(\Sigma, C, \mathcal{P}) = \{\, S \mid S \text{ is derived using an SOL-deduction from } \Sigma + C \text{ and } \mathcal{P}\}.$$

The *production from* $\Sigma + C$ *and* $\mathcal{P}$ is:

$$Prod(\Sigma, C, \mathcal{P}) = \mu\, \Delta(\Sigma, C, \mathcal{P})\,.$$

Note that $Newcarc(\Sigma, C, \mathcal{P})$ and $Prod(\Sigma, C, \mathcal{P})$ may contain an infinite number of clauses. To compute them practically, we have to restrict theories to a decidable subset of first-order logic and limit ourselves to finite domains.

The next proposition shows how the primitive $Newcarc(\Sigma, C, \mathcal{P})$ for a single clause C can be computed, by checking for each clause $S \in Prod(\Sigma, C, \mathcal{P})$ whether $\Sigma \models S$ or not. Typically, $Prod(\Sigma, C, \mathcal{P})$ contains considerably fewer clauses than $Carc(\Sigma \cup \{C\}, \mathcal{P})$ so that the approach outperforms incremental saturation procedures.

Proposition 4.6 Let C be a clause.

$$Newcarc(\Sigma, C, \mathcal{P}) = Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)\,.$$

Proof: By Theorem 4.3 (1), it is easy to see that

$$Prod(\Sigma, C, \mathcal{P}) \subseteq Th_{\mathcal{P}}(\Sigma \cup \{C\})\,.$$

Assume that $T \in Newcarc(\Sigma, C, \mathcal{P})$. Then, $T \in \mu[Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)]$. By Theorem 4.3 (2), there is a clause S in $Prod(\Sigma, C, \mathcal{P})$ such that S subsumes T. Since no clause in $Prod(\Sigma, C, \mathcal{P})$ properly subsumes T, T and S subsume each other so that they can be considered to be identical. Therefore, it holds that

$$Newcarc(\Sigma, C, \mathcal{P}) \subseteq Prod(\Sigma, C, \mathcal{P})\,.$$

It remains to be shown that $Prod(\Sigma, C, \mathcal{P}) - Newcarc(\Sigma, C, \mathcal{P}) \subseteq Th_{\mathcal{P}}(\Sigma)$. Suppose to the contrary, for $S \in Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)$, that $S \notin Newcarc(\Sigma, C, \mathcal{P})$. As $S \notin Th_{\mathcal{P}}(\Sigma)$, $S \notin Carc(\Sigma, \mathcal{P})$ holds. Because $S \in Prod(\Sigma, C, \mathcal{P})$, it must be that $S \in Th_{\mathcal{P}}(\Sigma \cup \{C\})$. Since $S \notin Carc(\Sigma \cup \{C\}, \mathcal{P})$ by the supposition and Proposition 3.7, there is a clause S' in $Carc(\Sigma \cup \{C\}, \mathcal{P})$ such that S' properly subsumes S. Since S' subsumes S and $S \notin Th_{\mathcal{P}}(\Sigma)$, clearly $S' \notin Th_{\mathcal{P}}(\Sigma)$. Thus, $S' \in Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)$. By Theorem 4.3 (2), there must be a clause S'' in $Prod(\Sigma, C, \mathcal{P})$ such that S'' subsumes S'. However, by Definition 4.5, $Prod(\Sigma, C, \mathcal{P})$ is closed under subsumption, contradiction. Hence, $S \in Newcarc(\Sigma, C, \mathcal{P})$. □

Proposition 4.6 says the primitive $Newcarc(\Sigma, C, \mathcal{P})$ is contained in the production from $\Sigma + C$ and $\mathcal{P}$. To remove theorems of Σ from the production, we have to test whether a clause S, produced from $\Sigma + C$ and $\mathcal{P}$, belongs to $Th_{\mathcal{P}}(\Sigma)$ or not. This test is a counterpart of checking the consistency of hypotheses with a theory in abduction and is undecidable in general. The question is how effectively this consistency checking can be performed in decidable cases. We already know that S belongs to $\mathcal{P}$. A direct implementation is to use the proof-finding property provided by Proposition 3.5: $\Sigma \models S$ if and only if $Prod(\Sigma, \neg S, \mathcal{P}) = \{\Box\}$. [6] In this case, since the only target clause produced from $\Sigma + \neg S$ is $\Box$, the production field $\mathcal{P}$ can be replaced with $\langle\, \emptyset \,\rangle$ so that **Skip** (Rule 6(a)i) will never be applied: there is a C-ordered linear refutation from $\Sigma \cup \{\neg S\}$ if and only if there is an SOL-deduction from $\Sigma + \neg S$ and $\langle\, \emptyset \,\rangle$ (see Corollary 4.24).

However, there is another way to check the derivability of S from Σ. When the characteristic literals $\mathbf{L}$ of $\mathcal{P}$ are small compared with the set of all literals $\mathcal{L}$, the computation of $Carc(\Sigma, \mathcal{P})$ can be performed more efficiently as the search focuses on $\mathcal{P}$. Having $Carc(\Sigma, \mathcal{P})$, consistency checking is much easier because the check can be reduced to subsumption tests on $Carc(\Sigma, \mathcal{P})$ by Proposition 3.5: $S \in Th_{\mathcal{P}}(\Sigma)$ if and only if there is a clause $T \in Carc(\Sigma, \mathcal{P})$ such that T subsumes S. We thus propose an intermediate approach between incremental saturation procedures based on Proposition 3.7 and an "interpreted" approach given by Proposition 4.6. It does not compute the saturated set $Carc(\Sigma \cup \{C\}, \mathcal{P})$, but computes and keeps another saturated set $Carc(\Sigma, \mathcal{P})$ for consistency checking. This kind of checking can be embedded into an SOL-deduction by adding the following rule into Definition 4.1:

4^+. For each $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$, P_i is not subsumed by any clause of $Carc(\Sigma, \mathcal{P})$.

This rule enables SOL-resolution to check the derivability from Σ of each partial clause P_i consisting of the literals that have been skipped so far. [7]

Let us denote by $\Delta_+(\Sigma, C, \mathcal{P})$ the clauses derived by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$ in which Rule 4^+ is incorporated. The *+-production from* $\Sigma + C$ *and*

[6] Here, we assume that Σ is satisfiable.

[7] For this purpose, it is efficient to keep $Carc(\Sigma, \mathcal{P})$ if it is much smaller than $Carc(\Sigma \cup \{C\}, \mathcal{P})$, although this depends on Σ and C (for example, if $\Sigma \cup \{C\}$ is inconsistent then $Carc(\Sigma \cup \{C\}, \mathcal{P})$ contains only $\Box$). The role of $Carc(\Sigma, \mathcal{P})$ in this case is similar to the minimal nogoods in the ATMS. Instead of using a "compiled" approach, i.e., checking with $Carc(\Sigma, \mathcal{P})$, the check can be carried out in a demand driven manner. For P_i such that $\Sigma \not\models P_i$, if l is skipped and added to P_i, we may check whether $\Sigma \not\models P_i \cup \{l\}$ or not by failing to find a proof of l from $\Sigma \cup \{\neg P_i\}$. For Theorist [Poole *et al.*, 1987], Poole [1989] uses a Prolog meta-interpreter for incremental consistency checking, and Satter, Goodwin and Goebel [1990] utilize nogoods that have been found in previously failed consistency checks.

$\mathcal{P}$ is defined as:

$$Prod_+(\Sigma, C, \mathcal{P}) = \mu\, \Delta_+(\Sigma, C, \mathcal{P})\,.$$

Proposition 4.7 $Prod_+(\Sigma, C, \mathcal{P}) = Newcarc(\Sigma, C, \mathcal{P})\,.$

Proof: Let $D_0, \ldots, D_n$ be an SOL-deduction, and $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$ and $D_j = \langle\, P_j,\ \vec{Q}_j \,\rangle$ its two structured clauses where $0 \leq i < j \leq n$. Suppose that P_i is subsumed by a clause $T_i \in Carc(\Sigma, \mathcal{P})$. Since P_i belongs to $\mathcal{P}$, $\Sigma \models P_i$ by Proposition 3.5. On the other hand, since there is a substitution θ such that $P_i\theta \subseteq P_j$, it holds that $\Sigma \models P_j$. By Proposition 3.5, there is a clause $T_j \in Carc(\Sigma, \mathcal{P})$ such that T_j subsumes P_j. The incorporation of Rule 4^+ into SOL-resolution thus prevents SOL-deductions producing clauses subsumed by some clauses from $Carc(\Sigma, \mathcal{P})$, but it does not affect the completeness result given in Theorem 4.3 (2) (since the completeness is not concerned with clauses belonging to $Th_{\mathcal{P}}(\Sigma)$). Hence, by Proposition 4.6, the present proposition follows. □

For a CNF formula F, $Newcarc(\Sigma, F, \mathcal{P})$ can be computed incrementally by using a series of SOL-deductions as follows.

Proposition 4.8 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\, \bigcup_{i=1}^{m} Prod(\Sigma_i, C_i, \mathcal{P})\,] - Th_{\mathcal{P}}(\Sigma)\,,$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$.

Proof: By Proposition 3.8,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu\,[\, \bigcup_{i=1}^{m} Newcarc(\Sigma_i, C_i, \mathcal{P})\,]\,.$$

Take any union of two successive primitive $Newcarc$ operations in the above equation. By applying Proposition 4.6, we get the following equation for such a union ($1 \leq k \leq m-1$).

$$\begin{aligned}
& \mu\,[\, Newcarc(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Newcarc(\Sigma_k, C_k, \mathcal{P})\,] \\
= \;& \mu\,[\, (\, Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\})\,) \\
& \quad \cup (\, Prod(\Sigma_k, C_k, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma_k)\,)\,] \\
= \;& \mu\,[\, (\, Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})\,) \\
& \quad - (\, Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\}) - Prod(\Sigma_k, C_k, \mathcal{P})\,)
\end{aligned}$$

$$
\begin{aligned}
& - (Th_{\mathcal{P}}(\Sigma_k) - Prod(\Sigma_k \cup \{C_k\}, C_{k+1}, \mathcal{P})) \\
& - (Th_{\mathcal{P}}(\Sigma_k \cup \{C_k\}) \cap Th_{\mathcal{P}}(\Sigma_k))] \\
= \; & \mu [(Prod(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})) \\
& - (Th_{\mathcal{P}}(\Sigma_{k+1}) - Prod(\Sigma_k, C_k, \mathcal{P})) - Th_{\mathcal{P}}(\Sigma_k)] \\
= \; & \mu [Prod(\Sigma_{k+1}, C_{k+1}, \mathcal{P}) \cup Prod(\Sigma_k, C_k, \mathcal{P})] - Th_{\mathcal{P}}(\Sigma_k) .
\end{aligned}
$$

The above equation can be used successively and extended to prove the proposition. □

As in the case of Proposition 4.7 for the primitive $Newcarc$ operation, the consistency checking for the new characteristic clauses of a CNF formula in Proposition 4.8 can be embedded into SOL-deductions by adding Rule 4^+.

Corollary 4.9 Let $F = C_1 \wedge \cdots \wedge C_m$ be a CNF formula. Then,

$$Newcarc(\Sigma, F, \mathcal{P}) = \mu [\bigcup_{i=1}^{m} Prod_+(\Sigma_i, C_i, \mathcal{P})] ,$$

where $\Sigma_1 = \Sigma$, and $\Sigma_{i+1} = \Sigma_i \cup \{C_i\}$, for $i = 1, \ldots, m-1$. □

As an application of computation of the new characteristic clauses, we now illustrate how SOL-resolution works for abduction. Recall that minimal explanations can be obtained by negating each clause in $Newcarc(\Sigma, C, \mathcal{P})$, where Σ is the theory, C is the negation of the query and $\mathcal{P}$ contains the negations of the hypotheses (Proposition 3.11). Now, we can assume that Σ does not include the newly added clause C. Then, in the ground case, C is used only as the top clause of each SOL-deduction, and does not have to be used elsewhere. This means that **Resolve** (Rule 6(a)ii in Definition 4.1) always takes a side clause from Σ, and that C stands for a single ground clause in a ground SOL-deduction.

In the general case, however, the nonground top clause C must be allowed to be used as a side clause to ensure the completeness of SOL-resolution. When it is used as side clauses with different instances, the single general clause C represents multiple ground clauses. Adding the general clause C to Σ is like adding an infinite number of ground instances of it. This situation leads to an *indefinite answer* to a query in abduction [Poole, 1991], and is analogous to (non-abductive) query-answering based on Model Elimination where disjunctive answers can occur only if the query is used more than once during a refutation [Wakayama and Payne, 1988]. In other words, to get all *definite answers*, it is sufficient to consider the case such that C is used only as the top clauses of SOL-deductions.

Example 4.10 Suppose that the theory Σ consists of the following two clauses:

$$\begin{array}{ll} (1) & \neg P(x) \vee Q(y,y) \vee R(z,x)\,, \\ (2) & \neg Q(x,y) \vee R(x,y)\,. \end{array}$$

Suppose also that the set of hypotheses is given as

$$\Gamma = \{P\}^+\,,$$

that is, the positive occurrences of the predicate P. We can set the production field to $\mathcal{P} = \langle\, \overline{\Gamma}\,\rangle = \langle\, \{P\}^-\,\rangle$. Now, consider the query,

$$G = \exists\, x\ (\, R(A,x)\,)\,.$$

The first SOL-deduction from $\Sigma + \neg G$ and $\mathcal{P}$ is as follows:

$$\begin{array}{lll} (3) & \langle\, \Box\,, & \underline{\neg R(A,x)}\,\rangle\,, \qquad \text{top clause} \\ (4) & \langle\, \Box\,, & \underline{\neg P(x)} \vee Q(y,y) \vee \boxed{\neg R(A,x)}\,\rangle\,, \qquad \text{resolution with (1)} \\ (5) & \langle\, \neg P(x)\,, & \underline{Q(y,y)} \vee \boxed{\neg R(A,x)}\,\rangle\,, \qquad \text{skip} \\ (6) & \langle\, \neg P(x)\,, & \underline{R(y,y)} \vee \boxed{Q(y,y)} \vee \boxed{\neg R(A,x)}\,\rangle\,, \qquad \text{resolution with (2)} \\ (7a) & \langle\, \neg P(A)\,, & \boxed{Q(A,A)} \vee \boxed{\neg R(A,A)}\,\rangle\,, \qquad \text{ancestry} \\ (7b) & \langle\, \neg P(A)\,, & \Box\,\rangle\,. \qquad \text{truncation} \end{array}$$

In the above SOL-deduction, since $\neg G$ is used only as the top clause, $P(A)$ is an explanation of the definite answer $R(A,A)$ from (Σ, Γ). Namely,

$$\Sigma \models P(A) \supset R(A,A)\,.$$

The second SOL-deduction from $\Sigma + \neg G$ and $\mathcal{P}$ takes the same four steps as the above (3)–(6), but instead of applying ancestry at (7), $R(y,y)$ is resolved upon against the clause $\neg R(A,x')$, yielding

$$\begin{array}{lll} (7a') & \langle\, \neg P(x)\,, & \boxed{R(A,A)} \vee \boxed{Q(A,A)} \vee \boxed{\neg R(A,x)}\,\rangle\,, \\ (7b') & \langle\, \neg P(x)\,, & \Box\,\rangle\,. \end{array}$$

In this case, $\neg G$ is used twice in the SOL-deduction. We get $\neg P(x)$, but $P(x)$ is not an explanation of any definite answer. This result just says that for any term t, $P(t)$ is an explanation of the indefinite answer $R(A,t) \vee R(A,A)$. Namely,

$$\Sigma \models \forall\, x\ (\, P(x) \supset R(A,x) \vee R(A,A)\,)\,.$$

4.2.3 Computing Characteristic Clauses

For computing the characteristic clauses $Carc(\Sigma, \mathcal{P})$, it is not necessary to check whether the clauses produced by using SOL-deduction are not theorems of Σ, unlike the cases of Propositions 4.6 and 4.8.

Proposition 4.11 The characteristic clauses with respect to $\mathcal{P} = \langle \mathbf{L}, Cond \rangle$ can be generated incrementally as: [8]

$$\begin{aligned} Carc(\emptyset, \mathcal{P}) &= \mu\{\, T \mid T \in Taut(\mathbf{L}) \text{ and } T \text{ satisfies } Cond \,\}, \\ Carc(\Sigma \cup \{C\}, \mathcal{P}) &= \mu[\, Carc(\Sigma, \mathcal{P}) \cup Prod(\Sigma, C, \mathcal{P}) \,]. \end{aligned}$$

Proof: The first equation is the same as Proposition 3.9. Now,

$$\begin{aligned} & Carc(\Sigma \cup \{C\}, \mathcal{P}) \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup Newcarc(\Sigma, C, \mathcal{P}) \,] \text{ (by Proposition 3.9)} \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup (Prod(\Sigma, C, \mathcal{P}) - Th_{\mathcal{P}}(\Sigma)) \,] \text{ (by Proposition 4.6)} \\ = \; & \mu[\, Carc(\Sigma, \mathcal{P}) \cup Prod(\Sigma, C, \mathcal{P}) \,]. \end{aligned}$$

Hence, the proposition. □

We should note that SOL-resolution was devised originally for efficient computation of the new characteristic clauses ($Newcarc$). Proposition 4.11 requires m separate productions for computing the characteristic clauses ($Carc$) of Σ if Σ contains m clauses, one for each of the clauses in Σ. If some other (non-incremental) process would compute all the $Carc$ in one production by taking all the clauses of Σ as input, then it might be superior. On the other hand, our method based on SOL-resolution is beneficial in particular to *incremental* computation of $Carc$. The above proposition thus illustrates the wide applicability of SOL-deductions; SOL-resolution is effective for computing $Newcarc$, and is general enough to be used for computing $Carc$.

4.2.4 Computing the CMS/ATMS

As an application of computation of the (new) characteristic clauses based on SOL-resolution, we now show how the CMS/ATMS [Reiter and de Kleer, 1987] characterized in Section 3.3.3 can be computed when the given theory is propositional.

[8] Notice that valid clauses from $Taut(\mathbf{L})$ decrease monotonically in the incremental construction of $Carc(\Sigma, \mathcal{P})$. In practice, $Carc(\emptyset, \mathcal{P})$ can be computationally replaced with $\emptyset$ though it may be infinite in the general case. Since no tautology will take part in any deduction (see Rule 3 in Definition 4.1), the same argument can be applied for $PI(\emptyset)$ in Proposition 4.12.

We should note that, in spite of its usefulness in a wide range of applications, the algorithms for the ATMS in [de Kleer, 1986; de Kleer, 1988] have not yet been proved to be correct with respect to the declarative semantics. Although the CMS is well defined and the basic connection between the resolution principle by Robinson [1965] and the CMS processing is given in [Reiter and de Kleer, 1987], there has not yet been any complete algorithm for computing labels of a formula for non-Horn theories using more popular and more sophisticated resolution strategies. Using SOL-resolution, we provide a sound and complete procedure of the CMS/ATMS for both label generating (the interpreted approach) and label updating (the compiled approach).

Interpreted Approach for a CMS

Recall that, in the CMS, the minimal supports for a formula F with respect to a theory Σ can be represented as:

$$\begin{aligned} MS(\Sigma, F) &= Newcarc(\Sigma, \neg F, \mathcal{P}_{\mathcal{L}}) \\ &= PI(\Sigma \cup \{\neg F\}) - PI(\Sigma) \end{aligned}$$

by Propositions 3.17 and 3.7. By using Proposition 4.6, we can generate the new characteristic clauses $Newcarc(\Sigma, C, \mathcal{P}_{\mathcal{L}})$ for a clause C without knowing the saturated sets, $PI(\Sigma)$ and $PI(\Sigma \cup \{C\})$. When F is a disjunctive normal form query, $MS(\Sigma, \neg F)$ can be computed using Proposition 4.8. Therefore, computation using Proposition 3.17 and Proposition 4.6 represents the *interpreted approach*. Note that in [Reiter and de Kleer, 1987] there is no description of an algorithm for the interpreted approach.

Compiled Approach for a CMS

Recall that the *compiled approach* to the CMS takes $PI(\Sigma)$ as input to find $MS(\Sigma, C)$ for any clause C, as shown in Proposition 3.19:

$$MS(\Sigma, C) = \mu \{ P - C \mid P \in PI(\Sigma) \text{ and } P \cap C \neq \emptyset \} .$$

Since the compiled approach keeps $PI(\Sigma)$, its updating cost is very high. However, for both purposes, constructing the prime implicates and updating them, the next proposition can be used:

Proposition 4.12 Given $PI(\Sigma)$ and a clause C, $PI(\Sigma \cup \{C\})$ can be found incrementally:

$$\begin{aligned} PI(\emptyset) &= \{ \alpha \vee \neg\alpha \mid \alpha \in \mathcal{A} \} , \text{ and} \\ PI(\Sigma \cup \{C\}) &= \mu [PI(\Sigma) \cup Prod(PI(\Sigma), C, \mathcal{P}_{\mathcal{L}})] . \end{aligned}$$

Proof: The proposition follows by setting $\mathcal{P}$ to $\mathcal{P}_{\mathcal{L}}$ in Proposition 4.11 and by using Proposition 4.6 and Lemma 3.15. □

By Proposition 4.12, the prime implicates can be incrementally constructed using every clause as a top clause. Thus the transmitted clauses Σ can be substituted for $PI(\Sigma)$. When a clause C is newly added, we just need to add the theorems derived from $PI(\Sigma)$ with top clause C and remove the subsumed clauses.

The computation of all prime implicates of Σ by Proposition 4.12 is much more efficient than the brute-force resolution method proposed briefly in [Reiter and de Kleer, 1987], which makes every possible resolution until no more unsubsumed clauses are produced. The computational superiority of the proposed technique as compared with a brute-force, saturation algorithm comes from the restriction of resolution. The key problem here is to generate as few subsumed clauses as possible, thus making as few subsumption tests as possible. Also, since our technique uses C-ordered linear resolution, it has naturally more restriction strategies than Kean and Tsiknis's [1990] extension of the consensus method [Tison, 1967] for generating prime implicates by set-of-support resolution.

This difference becomes larger when there are some distinguished literals representing assumptions in the ATMS case. The most important difference lies in the fact that the formulations by [Reiter and de Kleer, 1987; Kean and Tsiknis, 1990] require the computation of all prime implicates whereas the methods we describe below need only certain characteristic clauses that are a subset of the prime implicates, again avoiding "generate-and-test" manners [Inoue, 1990b]. We now make a detailed analysis of such computational properties for ATMSs.

Interpreted Approach for an ATMS

In Definition 3.23, we have defined an ATMS as a triple $\langle N, \Gamma, \Sigma \rangle$, where $N \subseteq \mathcal{L}$ (nodes), $\Gamma \subseteq N$ (assumptions), and Σ is a set of clauses (justifications). The label of $n \in N$ is:

$$Label(n, \Gamma, \Sigma) = \overline{Newcarc(\Sigma, \neg n, \mathcal{P})}, \quad \text{where } \mathcal{P} = \langle \overline{\Gamma} \rangle.$$

In the same way as the CMS, we will consider the following two problems, that is, abduction and saturation, concerning the computation of the labels of the nodes with respect to an ATMS:

1. *Generating labels.* Given an ATMS $\langle N, \Gamma, \Sigma \rangle$, compute $Label(n, \Gamma, \Sigma)$ for some node $n \in N$ from the original set Σ. This corresponds to the interpreted approach of the CMS.

2. *Updating labels*. Given an ATMS $\langle N, \Gamma, \Sigma \rangle$, the current label $Label(n, \Gamma, \Sigma)$ of each $n \in N$, and a newly added clause C, compute the new label $Label(n, \Gamma, \Sigma \cup \{C\})$ of every $n \in N$ with respect to $\langle N, \Gamma, \Sigma \cup \{C\} \rangle$. This corresponds to the compiled approach of the CMS.

Generating the label $Label(n, \Gamma, \Sigma)$ of a node n is straightforward by Proposition 4.6 and Definition 3.23. Moreover, a query is not restricted to being a literal of N in this case: for a general formula, Proposition 4.8 can be applied by converting it to CNF. Alternatively, Proposition 4.7 and Corollary 4.9 can be used because $Carc(\Sigma, \mathcal{P})$ (*minimal nogoods*) in this case can be computed more easily than $PI(\Sigma)$ in the CMS case.

Compiled Approach for an ATMS

In the compiled approach to an ATMS, the following result corresponding to Corollary 3.20 for the CMS and to a generalization of [Reiter and de Kleer, 1987, Theorem 7] holds:

Proposition 4.13 Let $\langle N, \Gamma, \Sigma \rangle$ be an ATMS, $n \in N$ a literal, and $\mathcal{P} = \langle \overline{\Gamma} \rangle$.

$$Newcarc(\Sigma, \neg n, \mathcal{P}) = \{\, P - \{n\} \mid P \in PI(\Sigma),\ n \in P \text{ and } P - \{n\} \text{ belongs to } \mathcal{P} \,\}.$$

Proof: ($\supseteq$) Let $P \in PI(\Sigma)$ such that $n \in P$ and $P - \{n\}$ belongs to $\mathcal{P}$. Then, since $P - \{n\} \subset P$, it holds that $\Sigma \not\models P - \{n\}$. Since $n \in P$ and $\Sigma \models P$, $\Sigma \cup \{\neg n\} \models P - \{n\}$. Therefore, $P - \{n\} \in Th_{\mathcal{P}}(\Sigma \cup \{\neg n\}) - Th_{\mathcal{P}}(\Sigma)$. As $P \in PI(\Sigma)$ and $n \in P$, for any clause $S \subset P - \{n\}$, $\Sigma \not\models S \cup \{n\}$, and thus $\Sigma \cup \{\neg n\} \not\models S$ holds. This implies that $P - \{n\} \in Carc(\Sigma \cup \{\neg n\}, \mathcal{P})$, and thus $P - \{n\} \in Newcarc(\Sigma, \neg n, \mathcal{P})$ (by Proposition 3.7). [9]

($\subseteq$) Let $S \in Newcarc(\Sigma, \neg n, \mathcal{P})$. As $\Sigma \cup \{\neg n\} \models S$, $\Sigma \models S \cup \{n\}$ holds. Suppose that there is a clause T in $Th(\Sigma)$ such that $T \subset S \cup \{n\}$ and that $T - \{n\}$ belongs to $\mathcal{P}$. Clearly, $\Sigma \models T \cup \{n\}$. Now for any clause S' such that $S' \subset S$, since $\Sigma \cup \{\neg n\} \not\models S'$, $\Sigma \not\models S' \cup \{n\}$ holds. Therefore, $S \subseteq T \subset S \cup \{n\}$. As n is a literal, $T = S$. However, $S \notin Carc(\Sigma, \mathcal{P})$, contradiction. Hence, $S \cup \{n\} \in PI(\Sigma)$. Replacing $S \cup \{n\}$ with P, we get the proposition. □

[9] Note that in this direction of the proof, n need not be a literal in N; the relation holds for a clause C. Namely, if $P \in PI(\Sigma)$, $C \subseteq P$, and $P - C$ belongs to $\mathcal{P}$, then $\neg(P - C)$ is a minimal explanation of C from (Σ, Γ). This result corresponds to a generalization of [Reiter and de Kleer, 1987, Theorem 3] for a general $\mathcal{P}$.

Proposition 4.13 shows that we can compute the label of a node from the prime implicates of Σ. Therefore, an approach may keep $PI(\Sigma)$ and when a new clause C is added we compute $PI(\Sigma \cup \{C\})$ by Proposition 4.12 for updating labels of nodes. However, compared with the CMS, many of the prime implicates are not significant for the task of an ATMS when the assumptions Γ are relatively small, although their computation is extremely high. In such a case, we do not want to compute all the prime implicates. Fortunately, we can compute a subset of $PI(\Sigma)$ enough to give labels by using the following stable production field:

Definition 4.14 Given an ATMS $\langle N, \Gamma, \Sigma \rangle$ and a production field $\mathcal{P} = \langle \overline{\Gamma} \rangle$, the production field $\mathcal{P}^*$ is defined as:

$$\mathcal{P}^* = \langle \overline{\Gamma} \cup N, \textit{ the number of literals from } N - \overline{\Gamma} \textit{ is at most one} \rangle.$$

Since $\mathcal{P}^*$ is stable, $Carc(\Sigma, \mathcal{P}^*)$ can be constructed incrementally by using Proposition 4.11:

$$Carc(\Sigma \cup \{C\}, \mathcal{P}^*) = \mu\,[\, Carc(\Sigma, \mathcal{P}^*) \cup Prod(\Sigma, C, \mathcal{P}^*)\,].$$

Here, we only need to keep Σ and $Carc(\Sigma, \mathcal{P}^*)$ for computing $Carc(\Sigma \cup \{C\}, \mathcal{P}^*)$. Now, looking further at Definition 4.14, the relationship between $Carc(\Sigma, \mathcal{P}^*)$ and $Carc(\Sigma, \mathcal{P})$ can be given exactly in the next lemma:

Lemma 4.15

$$\begin{aligned} & Carc(\Sigma, \mathcal{P}^*) \\ = \; & Carc(\Sigma, \mathcal{P}) \cup \{\, S \cup \{n\} \mid n \in N - \overline{\Gamma} \text{ and } S \in Newcarc(\Sigma, \neg n, \mathcal{P}) \,\}. \end{aligned}$$

Proof: $Carc(\Sigma, \mathcal{P}^*)$ can be divided into two disjoint sets of clauses: (1) containing no literal from $N - \overline{\Gamma}$, and (2) containing exactly one literal in $N - \overline{\Gamma}$. The former is exactly $Carc(\Sigma, \mathcal{P})$. Assume that C belongs to the latter set, and that $n \in C$ is a literal from $N - \overline{\Gamma}$. Then, $C - \{n\}$ contains only literals in $\overline{\Gamma}$ and thus belongs to $\mathcal{P}$. We must show that $C - \{n\} \in Carc(\Sigma \cup \{\neg n\}, \mathcal{P})$. Since $C \in Carc(\Sigma, \mathcal{P}^*)$, $\Sigma \models C$ and thus $\Sigma \cup \{\neg n\} \models C - \{n\}$. Suppose to the contrary that $Th_{\mathcal{P}}(\Sigma \cup \{\neg n\})$ contains a clause S such that $S \subset C - \{n\}$. Then, $S \cup \{n\} \subset C$ and $S \cup \{n\} \notin Th_{\mathcal{P}^*}(\Sigma)$ by the minimality of $C \in Th_{\mathcal{P}^*}(\Sigma)$. Since $\Sigma \not\models S \cup \{n\}$, $\Sigma \cup \{\neg n\} \not\models S$, contradiction. Therefore, $C - \{n\} \in Carc(\Sigma \cup \{\neg n\}, \mathcal{P})$. Since $C \in Carc(\Sigma, \mathcal{P}^*)$, obviously $C - \{n\} \notin Carc(\Sigma, \mathcal{P})$ holds. Hence, the lemma. □

Recall that the characteristic clauses $Carc(\Sigma, \langle\, \overline{\Gamma}\,\rangle)$ are called the *minimal* (or unsubsumed) *nogoods* in the ATMS terminology. Therefore, the knowledge base can consist of the justifications Σ, unsubsumed nogoods $Carc(\Sigma, \mathcal{P})$, and prime implicates in the form of a disjunction of a node n and the negation S of an element of its label. No other prime implicates are necessary. Having $Carc(\Sigma, \mathcal{P}^*)$, we can find the label of each node easily as follows:

Proposition 4.16 Let $\langle\, N, \Gamma, \Sigma\,\rangle$ be an ATMS, $n \in N$, $\mathcal{P} = \langle\, \overline{\Gamma}\,\rangle$, and $\mathcal{P}^*$ be the same as in Definition 4.14.

$$Newcarc(\Sigma, \neg n, \mathcal{P}) = \begin{cases} \{\, S - \{n\} \mid S \in Carc(\Sigma, \mathcal{P}^*), \text{ and } n \in S \,\} & \text{if } n \in N - \overline{\Gamma} \\ \{\, S - \{n\} \mid S \in Carc(\Sigma, \mathcal{P}), \text{ and } n \in S \,\} & \text{if } n \in N \cap \overline{\Gamma} \end{cases}$$

Proof: ($\supseteq$) Obvious from Proposition 4.13 and Lemma 4.15.

($\subseteq$) Let $T \in Newcarc(\Sigma, \neg n, \mathcal{P})$. By Proposition 4.13, $T \cup \{n\} \in PI(\Sigma)$.

(1) If $n \in N - \overline{\Gamma}$, then $T \cup \{n\}$ belongs to $\mathcal{P}^*$ because T belongs to $\mathcal{P}$. Therefore, $T \cup \{n\} \in Carc(\Sigma, \mathcal{P}^*)$.

(2) If $n \in N \cap \overline{\Gamma}$, then $T \cup \{n\}$ belongs to $\mathcal{P}$. Therefore, $T \cup \{n\} \in Carc(\Sigma, \mathcal{P})$. □

For updating the knowledge base when a new clause C is added, again we just compute $Carc(\Sigma \cup \{C\}, \mathcal{P}^*)$ from the previous $Carc(\Sigma, \mathcal{P}^*)$ incrementally by using Proposition 4.11. Since this computation guarantees the completeness of characteristic-clause-finding, the four properties of the ATMS labels in Proposition 3.24 are also satisfied in this case. Note that the μ operation removes all the previous prime implicates that are subsumed by some newly added prime implicates. This operation is also crucial to guarantee the label consistency because implicates subsumed by some nogood must be removed.

Example 4.17 Suppose that an ATMS is $\langle\, \{P, Q, U, V\},\ \{U, V\},\ \Sigma\,\rangle$ where

$$\Sigma = \{\, P \vee Q,\ \ \neg V \vee P \,\}.$$

In this case,

$$Carc(\Sigma, \mathcal{P}^*) = \Sigma \cup \{\, \neg U \vee U,\ \neg V \vee V \,\}.$$

Now suppose that a new clause,

$$\neg U \vee \neg P\,,$$

is added to Σ. Then the updating algorithm will find Q's new label U, as well as a new unsubsumed nogood $\neg U \vee \neg V$:

$$\begin{array}{l} \langle\ \Box,\ \ \underline{\neg U} \vee \neg P\ \rangle, \\ \langle\ \neg U,\ \ \underline{\neg P}\ \rangle, \end{array}$$

$$\swarrow \qquad\qquad \searrow$$

$$\begin{array}{rl} \langle\ \neg U,\ & \underline{Q} \vee \boxed{\neg P}\ \rangle, \\ \langle\ \neg U \vee Q,\ & \boxed{\neg P}\ \rangle, \\ \langle\ \neg U \vee Q,\ & \Box\ \rangle. \end{array} \qquad\qquad \begin{array}{rl} \langle\ \neg U,\ & \underline{\neg V} \vee \boxed{\neg P}\ \rangle, \\ \langle\ \neg U \vee \neg V,\ & \boxed{\neg P}\ \rangle, \\ \langle\ \neg U \vee \neg V,\ & \Box\ \rangle. \end{array}$$

We thus see that $Carc(\Sigma, \mathcal{P}^*)$ can be used to give labels for nodes. To maximize efficiency, however, $Carc(\Sigma, \mathcal{P}^*)$ can be kept as lemmas so that it can also be used for caching the result of the production; Namely, it can be utilized later as the bypass of steps of resolution that has been previously computed. The updating algorithm can be modified for this purpose and still establish the label completeness for various ATMSs [de Kleer, 1986; Reiter and de Kleer, 1987; de Kleer, 1988; de Kleer, 1989; Dressler, 1989]. See [Inoue, 1990a] for details and for the correspondence of the modified algorithm with de Kleer's [1986; 1988] label updating algorithms.

4.3 Variations

In the basic procedure in Section 4.2.1, no priority is given among the two rules, **Skip** (Rule 6(a)i) and **Resolve** (Rule 6(a)ii), in Step 6a of an SOL-deduction (Definition 4.1). That is, these two rules are treated as alternatives. This treatment is necessary to ensure the completeness of SOL-resolution. In this section, we violate this requirement, and show properties of efficient variations of SOL-resolution and their application to AI problems. Note that the **Reduce** rule (Rule 6(a)iii) still remains as an alternative choice to the other two rules (see Remark (4) of Definition 4.1).

4.3.1 Preferring Resolution

The first variation, *SOL-R deduction*, makes **Resolve** precede **Skip**. Namely, **Skip** is attempted only when **Resolve** cannot be applied.

In a special case of SOL-R deductions where the literals are not distinguished, that is, the production field is fixed to $\mathcal{P}_{\mathcal{L}}$, **Skip** is always applied whenever **Resolve** cannot be applied for any selected literal in a deduction. In abduction, the resultant procedure in this case "hypothesizes whatever cannot be proven". Many resolution-based abductive systems applied to diagnosis have favored this sort of *most-specific* explanation [Stickel, 1989]. This is also called *dead-end* abduction and was first

proposed by Pople [1973] in his abductive procedure based on SL-resolution. [10] The criterion is also used by Cox and Pietrzykowski [1986].

Note that SOL-R resolution used as an abductive procedure is not complete for finding even the most-specific explanations. Like other procedures [Pople, 1973; Cox and Pietrzykowski, 1986], it has an undesirable feature of possibly yielding different results depending on different literal ordering in each clause. For example, if the center clause is $P(x) \vee Q(x)$ and $P(x)$ is solved with $x = A$, then the instantiation may force $Q(A)$ to be skipped for lack of resolution possibilities. However, if $Q(x)$ is considered first without being instantiated, it might be resolvable and thus might not be skipped. Dead-end abduction is thus incompatible with C-ordering [Stickel, 1989].

4.3.2 Preferring Skip

In the next variation, *SOL-S deduction*, **Skip** and **Resolve** are placed in Step 6a of SOL-deductions in the reverse order of SOL-R deductions. That is, when the selected literal belongs to $\mathcal{P}$, only **Skip** is applied by ignoring the possibility of **Resolve**:

6(a)i-6(a)ii$\star$. **(Skip & Cut)**
If $P_i \cup \{l\}$ belongs to $\mathcal{P}$, then $P_{i+1} = P_i \cup \{l\}$ and $\vec{R_{i+1}}$ is the ordered clause obtained by removing l from $\vec{Q}_i$.
Otherwise, apply the **Resolve** rule (Rule 6(a)ii).

6(a)iii$\star$. **(Reduce)** The same as Rule 6(a)iii.

This skip-preference has the following nice properties. Firstly, it enables the procedure to prune the branch of the search tree that would have resulted from the literal being resolved upon. Secondly, SOL-S deductions are correct model-theoretically in the following sense. Let us divide the set of clauses $\Delta(\Sigma, C, \mathcal{P})$ produced by using SOL-deductions from $\Sigma + C$ and $\mathcal{P}$, not necessarily closed under subsumption, into two sets, say Δ_1 and Δ_2, such that

$$\Delta(\Sigma, C, \mathcal{P}) = \Delta_1 \cup \Delta_2 \text{ and } \Sigma \cup \Delta_1 \models \Delta_2 .$$

Recall that $Newcarc(\Sigma, C, \mathcal{P}) \subseteq Prod(\Sigma, C, \mathcal{P}) = \mu\Delta(\Sigma, C, \mathcal{P})$ (Definition 4.5). Then adding Δ_2 to Δ_1 never changes the models of $\Sigma \cup \Delta_1$:

$$Mod(\,\Sigma \cup \Delta_1\,) = Mod(\,\Sigma \cup \Delta(\Sigma, C, \mathcal{P})\,) ,$$

where $Mod(\Sigma)$ is the models of Σ. Thus, only Δ_1 needs to be computed model-theoretically. The next theorem shows that SOL-S deductions produce precisely such a Δ_1 when C is not used as a side clause in them.

[10] Pople's *synthesis* operation performs "factor-and-skip".

Theorem 4.18 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which C is used only as the top clause, then there is a set δ of clauses each of which is derived by an SOL-S deduction from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models T$.

Proof: See Section 4.5 for the proof. □

Let us denote by $\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P})$ the clauses derived by using SOL-S deductions from $\Sigma + C$ and $\mathcal{P}$. The *S-production from* $\Sigma + C$ *and* $\mathcal{P}$ is defined as:

$$Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P}) = \mu\, \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P})\,.$$

Corollary 4.19 $\Sigma \cup Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models Prod(\Sigma, C, \mathcal{P})\,.$

Proof: It is obvious to see from Theorem 4.18 that

$$\Sigma \cup \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models \Delta(\Sigma, C, \mathcal{P})\,.$$

The corollary follows from the fact that

$$\mu\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \models \Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \;\text{ and }\; \mu\Delta(\Sigma, C, \mathcal{P}) \subseteq \Delta(\Sigma, C, \mathcal{P}).$$

□

Notice that $Prod_{\mathbf{S}}(\Sigma, C, \mathcal{P})$ is not always a subset of $Prod(\Sigma, C, \mathcal{P})$ although $\Delta_{\mathbf{S}}(\Sigma, C, \mathcal{P}) \subseteq \Delta(\Sigma, C, \mathcal{P})$ holds. Thus a clause in the S-production may not be a new characteristic clause.

Example 4.20 Suppose that the theory is

$$\Sigma = \{\, P \supset G,\quad Q \supset G,\quad P \vee Q \,\}\,,$$

and that $\mathcal{P} = \langle\, \{\neg P,\, \neg Q\} \,\rangle$. In this case,

$$Prod_{\mathbf{S}}(\Sigma, \neg G, \mathcal{P}) = \{\, \neg P,\ \neg Q \,\}\,.$$

However, if SOL-resolution is used, we see that

$$Prod(\Sigma, \neg G, \mathcal{P}) = \{\Box\}\,.$$

The empty clause cannot be produced by using SOL-S deductions, but Theorem 4.18 is verified:

$$\Sigma \cup \{\, \neg P,\ \neg Q \,\} \models \Box\,.$$

When SOL-S deductions are applied to abduction especially for natural language understanding, it is sometimes called *least-specific* abduction [Stickel, 1989]. In abduction, recall that E is an explanation of G from (Σ, Γ) if $\neg E \in Prod(\Sigma, \neg G, \mathcal{P})$ for $\mathcal{P} = \langle\, \overline{\Gamma}\, \rangle$ and $\Sigma \not\models \neg E$. Now, when seeking definite answers in the abductive framework, each clause of $\neg G$ is used only as the top clause of any SOL-deduction. If that is so, Corollary 4.19 says that for any clause $\neg E \in Prod(\Sigma, \neg G, \mathcal{P})$, there exists a set of clauses $\delta \subseteq Prod_{\mathbf{S}}(\Sigma, \neg G, \mathcal{P})$ such that $\Sigma \cup \delta \models \neg E$ holds. Thus, the explanations in $\overline{Prod_{\mathbf{S}}(\Sigma, \neg G, \mathcal{P})}$ are the *weakest* in the sense that for any explanation E of G, there exist explanations $E_1, \ldots, E_n$ in $\overline{Prod_{\mathbf{S}}(\Sigma, \neg G, \mathcal{P})}$ such that

$$\Sigma \cup \{E\} \models E_1 \vee \cdots \vee E_n\,.$$

On the contrary, since the CMS/ATMS are used for computing all and only *minimal* (in the sense of set inclusion of literals) supports for a query, they should apply SOL-resolution instead of using SOL-S deductions.

In circumscription, SOL-S deduction is particularly desirable since we want to answer whether a query holds in every minimal model or not; the purpose of using explanation-based procedures is purely model-theoretic. One of the advantages of Przymusinski's [1989] procedure lies in the fact that MILO-resolution performs a kind of SOL-S deduction [Inoue and Helft, 1990]. See Section 5.3.1 in the next chapter for details. Similarly, SLINF-resolution [Minker and Rajasekar, 1990] prefers **Skip** to **Resolve** in deriving positive ground clauses from the axioms and the top clause. [11]

4.3.3 Between Skipping and Resolving

We can consider another interesting procedure that offers an intermediate alternative to the SOL-R and SOL-S deductions. The set of literals $\mathcal{L}$ can be divided into four sets: those never skipped, those skipped only if they cannot be resolved, those which can be non-deterministically chosen as either skipped or resolved, and those immediately skipped. This is useful for the sort of abduction where we would like to get explanations with appropriate amount of detail.

One further generalization of this idea would be *best-first* abduction. Stickel [1989] uses the minimum-cost proof for Horn theories where we can choose an operation whose expected computational cost is minimum. Unfortunately, it is difficult to apply this idea to non-Horn theories.

[11] We should comment that Theorem 13 of [Minker and Rajasekar, 1990] can be regarded as a special case of our Corollary 4.19. However, its proof lacks a careful analysis of the completeness of SLI-resolution for consequence-finding and ignores the case where ancestry operations occur in an SLI-derivation of a clause that is not derived by an SLINF-derivation.

4.3.4 Approximation

We can consider more drastic variations of the basic procedure. To do so, let us remember the complexity issues of consequence-finding in the propositional case. [12] These have been examined recently for the CMS/ATMS in [Provan, 1990; Selman and Levesque, 1990]. Provan [1990] shows that the ATMS inherits complexity from the enumeration of prime implicates, which is NP-hard. Thus any complete algorithm for computing ATMS labels is intractable. Selman and Levesque [1990] show that finding *one* explanation of an atom from a Horn theory and a set of atomic hypotheses is NP-hard. Therefore, even if we *abandon the completeness* of deductions with respect to the primitive *Newcarc* operation, for instance, by limiting the production to only those clauses having some small number of literals belonging to $\mathcal{P} = \langle \overline{\Gamma},\ \textit{length is fewer than } k \rangle$, it is still intractable.

Is there any way to overcome the computational difficulty? We can consider approximation of abduction; either *discard the consistency* or *dispense with the soundness*. In the former case, we may only run an SOL-deduction and believe the result, omitting consistency checking described in Section 4.2.2. This sort of optimistic reasoner does not care about ramification. On the other hand, the latter case happens if we skip literals not belonging to the characteristic literals: the soundness is violated in the sense that there is a clause S produced from $\Sigma + C$ and $\mathcal{P}$ such that $S \notin Th_{\mathcal{P}}(\Sigma \cup \{C\})$. [13] This is an extreme case of the SOL-S deductions in Section 4.3.2. We can stop deductions in accordance with computational resources; the unresolved literals in a leaf of the deduction are then immediately skipped. These skipped literals are dealt with as *defaults* and will be reconsidered later. Levesque [1989] also gives a hint for this kind of computation in terms of *explicit* abduction.

The fact that the procedure is sound and complete is valuable although it takes exponential time to get the desired answer. This is because we can improve the quality of solutions as time goes by; we can expect to get the correct answer if we can spend enough time in solving the problem. This property of being an "anytime algorithm", which, when resource limits are encountered, can stop and return the best conclusion so far derived, is a desirable feature for any future AI system.

---

[12] The complexity of consequence-finding in the general case is bounded by the limitation that the set of all theorems is recursively enumerable. Notice, however, that whether a given formula is *not* a logical consequence of such a theory cannot be necessarily determined in a finite amount of time.

[13] However, since for any structured clause $D_i = \langle P_i,\ \vec{Q}_i \rangle$ in any deduction from $\Sigma + C$ and $\mathcal{P}$, it holds that $\Sigma \cup \{C\} \models P_i \cup Q_i$, we can always guarantee that $S \in Th(\Sigma \cup \{C\})$.

4.4 Summary

In the last and the present chapters, we have revealed the importance of consequence-finding in AI techniques. In Chapter 3, we have provided a useful theoretical framework for many reasoning procedures involved with AI. Most advanced reasoning mechanisms such as abduction and default reasoning require global search in their proof procedures. This global character is strongly dependent on consequence-finding, in particular on those theorems of the theory belonging to production fields. That is why we need some complete procedure for consequence-finding.

For this purpose, we have provided an effective mechanism for implementing the consequence-finding framework. In this chapter, we have proposed SOL-resolution, an extension of C-ordered linear resolution augmented by the skip rule. The procedure is sound and complete for finding the new characteristic clauses. The significant innovation of the results presented is that the procedure is direct relative to the given production field. We have also presented incomplete but efficient variations of the basic procedures with different consequence-finding properties.

4.5 Proofs

In this section, we show the proofs of Theorems 4.3 and 4.18.

Firstly, we prove the soundness of SOL-resolution.

Lemma 4.21 Let $D_0, \ldots, D_n$ be an SOL-deduction of S from $\Sigma + C$ and $\mathcal{P}$. For each $D_i = \langle P_i, \vec{Q_i} \rangle$ $(0 \leq i \leq n-1)$, it holds that

$$\Sigma \cup \{ P_0 \cup Q_0, \ldots, P_i \cup Q_i \} \models P_{i+1} \cup Q_{i+1} .$$

Notice that $P_0 \cup Q_0 = C$ and $P_n \cup Q_n = S$.

> **Proof:** Since truncation (Rule 6b) does not change the clausal meaning of each structured clause, we can consider only Rule 6a. Let $D_{i+1} = \langle P_{i+1}, \vec{Q_{i+1}} \rangle$ $(0 \leq i \leq n-1)$ be the structured clause obtained from $D_i = \langle P_i, \vec{Q_i} \rangle$ by applying one of the three choices, **Skip** (Rule 6(a)i), **Resolve** (Rule 6(a)ii) or **Reduce** (Rule 6(a)iii), and truncation.
>
> 1. **Skip** is applied. In this case, $P_i \cup Q_i = P_{i+1} \cup Q_{i+1}$.
> 2. **Resolve** is applied. In this case, $\Sigma \cup \{C, P_i \cup Q_i\} \models P_{i+1} \cup Q_{i+1}$.
> 3. **Reduce** is applied. If factoring (Rule 6(a)iiiA) is applied, then $\{P_i \cup Q_i\} \models P_{i+1} \cup Q_{i+1}$. If ancestry (Rule 6(a)iiiB) is applied, then there exists a structured clause $D_j = \langle P_j, \vec{Q_j} \rangle$ $(j < i)$ such that $\{P_i \cup Q_i, P_j \cup Q_j\} \models P_{i+1} \cup Q_{i+1}$ (see [Chang and Lee, 1973, Lemma 7.2]).

By combining the above three cases, the lemma holds. □

Theorem 4.3 (1) (Soundness of SOL-Resolution)
If a clause S is derived using an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$, then S belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$.

Proof: Let $D_0, \ldots, D_n$ be an SOL-deduction of S from $\Sigma + C$ and $\mathcal{P}$. Since **Skip** is applied to only those literals belonging to $\mathcal{P}$ and Rule 5 is contained in the definition, it is easy to see that S belongs to $\mathcal{P}$.

We prove that $\Sigma \cup \{C\} \models S$. Take any model M of $\Sigma \cup \{C\}$. Since $\Sigma \cup \{C\} \models P_1 \cup Q_1$ by Lemma 4.21, M satisfies $P_1 \cup Q_1$ and is a model of $\Sigma \cup \{C, P_1 \cup Q_1\}$. Again by Lemma 4.21, M satisfies $P_2 \cup Q_2$, and further applications of Lemma 4.21 lead to the fact that M satisfies $P_n = S$. Hence, $S \in Th_{\mathcal{P}}(\Sigma \cup \{C\})$. □

Before giving the completeness proof of SOL-resolution for consequence-finding, we prove the completeness of SOL-resolution for proof-finding. In proof-finding, since the only target clause produced from $\Sigma + C$ is □, we can consider the production field $\mathcal{P}_\emptyset = \langle\, \emptyset \,\rangle$ so that **Skip** (Rule 6(a)i) will never be applied. Thus, for an SOL-deduction **D** of □ from $\Sigma + C$ and $\mathcal{P}_\emptyset$,

$$\langle\, \Box,\, \vec{C} \,\rangle, \ldots, \langle\, \Box,\, \vec{Q_i} \,\rangle, \ldots, \langle\, \Box,\, \Box \,\rangle\,,$$

we identify **D** with a sequence of ordered clauses,

$$\vec{C}, \ldots, \vec{Q_i}, \ldots, \Box\,.$$

We call SOL-resolution with the production field $\mathcal{P}_\emptyset$ *OL*-resolution.*

Definition 4.22 (OL*-Refutation) Given a set of clauses Σ, and a clause C, an *OL*-refutation from* $\Sigma + C$ consists of a sequence of ordered clauses, $\vec{Q_0}, \vec{Q_1}, \ldots, \vec{Q_n}$, such that:

1. $Q_0 = C$.
2. $Q_n = \Box$.
3. For each $\vec{Q_i}$, Q_i is not a tautology.
4. For each $\vec{Q_i}$, Q_i is not subsumed by any Q_j with the empty substitution, where $\vec{Q_j}$ is a previous ordered clause, $j < i$.

5. $\vec{Q_{i+1}}$ is generated from $\vec{Q_i}$ according to the following steps:
   (a) Let l be the left-most literal of $\vec{Q_i}$. $\vec{R_{i+1}}$ is obtained by applying one of the rules:
      i. **(Resolve0)** If there is a clause B_i in $\Sigma \cup \{C\}$ such that $\neg k \in B_i$ and l and k are unifiable with mgu θ, then $\vec{R_{i+1}}$ is an ordered clause obtained by concatenating $\vec{B_i}\theta$ and $\vec{Q_i}\theta$, framing $l\theta$, and removing $\neg k\theta$.
      ii. **(Reduce0)** If either
         A. $\vec{Q_i}$ contains an unframed literal k that is either different from l (*factoring*) or another occurrence of l (*merge*), or
         B. $\vec{Q_i}$ contains a framed literal $\boxed{\neg k}$ (*ancestry*),

      and l and k are unifiable with mgu θ, then $\vec{R_{i+1}}$ is obtained from $\vec{Q_i}\theta$ by deleting $l\theta$.
   (b) $\vec{Q_{i+1}}$ is obtained from $\vec{R_{i+1}}$ by deleting every framed literal not preceded by an unframed literal in the remainder (*truncation*).

We can see that the above definition of OL*-refutation is a variant of SL-refutation [Kowalski and Kuhner, 1971] or Model Elimination refutation [Loveland, 1978]. However, as noted in Remark (4) of Definition 4.1, OL*-refutation is different from OL-refutation [Chang and Lee, 1973] because ancestry (Rule 5(a)iiB) is an alternative choice to **Resolve** (Rule 5(a)i) or factoring (Rule 5(a)iiA). According to the completeness results for SL-refutation or Model Elimination refutation, it is not difficult to verify the completeness of OL*-refutation.

Theorem 4.23 If Σ is satisfiable and $\Sigma \cup \{C\}$ is unsatisfiable, then there exists an OL*-refutation from $\Sigma + C$. □

In SOL-resolution, since the **Skip** rule is defined as an alternative rule to others, a refutation can be obtained without using **Skip** as if the rule does not exist like OL*-refutation. Therefore, for any stable production field $\mathcal{P}$, we can identify an SOL-deduction of $\Box$ from $\Sigma + C$ and $\mathcal{P}$ with an OL*-refutation from $\Sigma + C$ as follows.

Corollary 4.24 Let $\mathcal{P}$ be any stable production field. If Σ is satisfiable and $\Sigma \cup \{C\}$ is unsatisfiable, then there exists an SOL-deduction of $\Box$ from $\Sigma + C$ and $\mathcal{P}$. □

The proof of the completeness of SOL-resolution for consequence-finding can be seen as an extension of the results for m.s.l. (merge, subsumtion, linear) resolution by Minicozzi and Reiter [1972, Theorems 1 and 2]. In addition to these techniques, we have to take C-ordering and the skip operation into account.

If Σ is a set of clauses and T is a clause, we write

$$\Sigma_T = \{\, C \mid C' \in \Sigma \text{ and } C = C' - T \,\}.$$

We first show the completeness proof for the ground case.

Lemma 4.25 [Minicozzi and Reiter, 1972, Lemma 1] If Σ is an unsatisfiable set of ground clauses and T is a ground clause, then Σ_T is unsatisfiable. □

Lemma 4.26 Let Σ be a set of ground clauses and T a ground clause. If Σ_T is unsatisfiable, then $\Sigma \models T$.

Proof: Assume to the contrary that $\Sigma \not\models T$. Since $\Sigma \cup \{\neg T\}$ is satisfiable, there is a model M of $\Sigma \cup \{\neg T\}$. Let C be any clause in Σ. Since M satisfies C and every $\neg t$ for any literal t of T, M satisfies $C - T$ and is a model of Σ_T. Hence, Σ_T is satisfiable, contradiction. □

Lemma 4.27 Let Σ be a set of ground clauses, and T and C ground clauses. If $\Sigma \not\models T$ and $\Sigma \cup \{C\} \models T$, then there is an OL*-refutation from $\Sigma_T + (C - T)$.

Proof: Suppose that $\Sigma \not\models T$. By Lemma 4.26, Σ_T is satisfiable.

Suppose further that $\Sigma \cup \{C\} \models T$. This implies that $\Sigma \cup \{C\} \cup \{\neg T\}$ is unsatisfiable. Since $\{\neg T\}_T = \{\neg T\}$, $\Sigma_T \cup \{C - T\} \cup \{\neg T\}$ is unsatisfiable by Lemma 4.25. However, no literal of the CNF formula $\neg T$ has the literal complementary to a literal in the clauses of $\Sigma_T \cup \{C - T\}$. Hence, $\Sigma_T \cup \{C - T\}$ is unsatisfiable.

By Theorem 4.23, there is an OL*-refutation from $\Sigma_T + (C - T)$. □

Theorem 4.28 Let Σ be a set of ground clauses, T and C ground clauses, and $\mathcal{P}$ a stable production field. If T does not belong to $Th_{\mathcal{P}}(\Sigma)$ but belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $S \subseteq T$.

Proof: Suppose that $T \in Th_{\mathcal{P}}(\Sigma \cup \{C\}) - Th_{\mathcal{P}}(\Sigma)$. Obviously, $\Sigma \not\models T$ and $\Sigma \cup \{C\} \models T$. By Lemma 4.27, there is an OL*-refutation,

$$\vec{Q_0}, \ldots, \vec{Q_i}, \ldots, \vec{Q_m},$$

from $\Sigma_T + (C - T)$. Note that $Q_0 = C - T$ and $Q_m = \Box$.

In this refutation, replace each occurrence of a clause $B_i \in \Sigma_T \cup \{C - T\}$ used in an application of **Resolve0** (Rule 5(a)i) to $\vec{Q_i}$ with the clause B_i' of $\Sigma \cup \{C\}$ from which it was derived. Now, construct a sequence **D** of structured clauses

$$D_{0,0}, \ldots, D_{0,k(0)}, \ldots, D_{i,0}, \ldots, D_{i,k(i)}, \ldots, D_{m,0}, \ldots, D_{m,k(m)},$$

from the refutation according to the following steps:

1. Let $D_{0,0} = \langle\, \Box,\, \vec{C}\,\rangle$.
2. $D_{i,j+1} = \langle\, P_{i,j+1},\, \vec{Q}_{i,j+1}\,\rangle$ $(0 \leq i \leq m,\ 0 \leq j \leq k(i) - 1)$ is obtained from $D_{i,j} = \langle\, P_{i,j},\, \vec{Q}_{i,j}\,\rangle$ in the following way:
   (a) If the left-most literal of $\vec{Q}_{i,j}$ is the same as the left-most literal of $\vec{Q}_i$ in the OL*-refutation, do not construct $D_{i,j+1}$. Set $k(i) = j$.
   (b) Otherwise, skip the left-most literal l of $\vec{Q}_{i,j}$, i.e., let $P_{i,j+1} = P_{i,j} \cup \{l\}$. $\vec{Q}_{i,j+1}$ is obtained from $\vec{Q}_{i,j}$ by removing the given occurrence of l, and truncating the remainder.
3. $D_{i+1,0} = \langle\, P_{i+1,0},\, \vec{Q}_{i+1,0}\,\rangle$ $(0 \leq i \leq m - 1)$ is constructed from $D_{i,k(i)} = \langle\, P_{i,k(i)},\, \vec{Q}_{i,k(i)}\,\rangle$ in the following way:
   (a) Let $P_{i+1,0} = P_{i,k(i)}$.
   (b) Let l be the left-most literal of $\vec{Q}_{i,k(i)}$. $\vec{Q}_{i+1,0}$ is obtained as follows.
      i. If **Resolve0** (Rule 5(a)i) was applied to $\vec{Q}_i$ in the OL*-refutation, deducing $\vec{Q}_{i+1}$ through truncation, then $\vec{Q}_{i+1,0}$ is the ordered clause obtained by concatenating $\vec{B}'_i$ and $\vec{Q}_{i,k(i)}$, framing l, removing $\neg l$, and truncating the remainder.
      ii. If **Reduce0** (Rule 5(a)ii) was applied to $\vec{Q}_i$ in the OL*-refutation, deducing $\vec{Q}_{i+1}$ through truncation, then $\vec{Q}_{i+1,0}$ is obtained from $\vec{Q}_{i,k(i)}$ by deleting the given occurrence of l, and truncating the remainder.

Note that the left-most literal of $\vec{Q}_{i,k(i)}$ $(1 \leq i \leq m)$ is always the same as the left-most literal of $\vec{Q}_i$ in the refutation. Notice also that $k(i)$ is the number of literals skipped in the sequence from $D_{i,0}$ to $D_{i+1,0}$, i.e.,

$$k(i) = |P_{i+1,0} - P_{i,0}|\,.$$

In other words, $P_{i+1,0} = P_{i,0} \cup S_i$, where S_i is the set of unframed literals in $\vec{Q}_{i,0}$ not preceded by the left-most literal of $\vec{Q}_{i+1}$.

Now, let $S = P_{m,k(m)}$. We see that $D_{m,k(m)} = \langle\, S,\, \Box\,\rangle$. In $\mathbf{D}$, since every literal in any $P_{i,j}$ is a literal of T which comes from $B'_i \cap T$ $(= B'_i - B_i)$ or $C \cap T$ $(= C - (C - T))$, we verify that $S \subseteq T$.

It remains to be verified that $\mathbf{D}$ is actually an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$.

1. In each construction of $D_{i+1,0}$ from $D_{i,k(i)}$ $(0 \leq i \leq m-1)$, the rules **Resolve0** and **Reduce0** have been changed to just the rules **Resolve** and **Reduce**.
2. In each construction of $D_{i,j+1}$ from $D_{i,j}$ $(0 \leq i \leq m,\ 0 \leq j \leq k(i)-1)$, **Skip** has been used. By the supposition, since T belongs to $\mathcal{P}$, every $P_{i,j}$ belongs to $\mathcal{P}$ and the condition for the application of **Skip** and Rule 5 are satisfied.
3. No clause of $B_i \in \Sigma_T \cup \{C-T\}$ contains a literal of $\neg T$ because each B_i is used in the OL*-refutation from $\Sigma_T + (C-T)$. This implies that no clause of $B'_i \in \Sigma \cup \{C\}$ contains the complement of a literal of T so that for any structured clause $D_{i,j} = \langle P_{i,j},\ \vec{Q_{i,j}} \rangle$, $P_{i,j} \cup Q_{i,j}$ is not a tautology.
4. Each $\vec{Q_{i,j}}$ in **D** contains all the literals of $\vec{Q_i}$ in the OL*-refutation. Furthermore, every literal in $Q_{i,j} - Q_i$ is a literal of T and thus does not appear in any previous Q_k $(k < i)$ in the refutation. Therefore, as Q_i is not subsumed by any Q_k $(k < i)$ with the empty substitution, $Q_{i,j}$ is not subsumed by any other $Q_{k,l}$ $(k \leq i, l \leq j)$ with the empty substitution.

□

We now lift the result of Theorem 4.28 to the general level. We use the technique, first used by Slagle, Chang and Lee [1969], of introducing new distinct constants into the Herbrand Universe.

Lemma 4.29 [Slagle *et al.*, 1969, Theorem 1] Let Σ be a set of clauses and T a ground clause. $\Sigma \models T$ if and only if there is a finite set Σ' of ground instances of clauses in Σ over the Herbrand Universe of $\Sigma \cup \{T\}$ such that $\Sigma' \models T$. □

Theorem 4.3 (2) (Completeness of SOL-Resolution)
If a clause T does not belong to $Th_{\mathcal{P}}(\Sigma)$, but belongs to $Th_{\mathcal{P}}(\Sigma \cup \{C\})$, then there is an SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S subsumes T.

Proof: Let $x_1, \ldots, x_m$ be all the individual variables of T. Let $b_1, \ldots, b_m$ be new distinct constants not appearing anywhere in Σ, C or T. Since $\Sigma \cup \{C\} \models T$, it holds that $\Sigma \cup \{C\} \models T\theta$ for a substitution $\theta = \{x_1/b_1, \ldots, x_m/b_m\}$ (i.e., b_i replaces x_i for $i = 1, \ldots, m$). Similarly, as $\Sigma \not\models T$, it must be that $\Sigma \not\models T\theta$. Let $H(b_1, \ldots, b_m)$ be the Herbrand Universe of $\Sigma \cup \{C, T\theta\}$. By Lemma 4.29, there exist two sets of ground clauses, Σ' and Σ'', satisfying the conditions:

1. $\Sigma' \not\models T\theta$, where Σ' is a finite set of ground instances of Σ over $H(b_1, \ldots, b_m)$, and
2. $\Sigma'' \models T\theta$, where Σ'' is the union of Σ' and a minimal set of ground instances of C over $H(b_1, \ldots, b_m)$.

By Theorem 4.28, for some ground clause $C' \in \Sigma'' - \Sigma'$, there is a ground SOL-deduction $\mathbf{D}'$ of S' from $(\Sigma'' - C') + C'$ and $\mathcal{P}$ such that $S' \subseteq T\theta$. In $\mathbf{D}'$, replace each occurrence of a clause $B_i' \in \Sigma''$ used in an application of **Resolve** with the clause B_i of $\Sigma \cup \{C\}$ of which it is a ground instance. Also, replace each structured clause D_i' in $\mathbf{D}'$ with the general structured clause D_i which is constructed from its parent (and the corresponding B_{i-1} if **Resolve** is applied). To make the number of skipped literals in P_i of $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$ always the same as the number in P_i' of the corresponding $D_i' = \langle\, P_i',\ \vec{Q}_i' \,\rangle$ in $\mathbf{D}'$, factoring should be inserted between two successive structured clauses whenever it is necessary, and merge operations should be replaced with factoring and reordered appropriately. The resultant sequence $\mathbf{D}$ is a general SOL-deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that S has, as an instance, S'. So, there is a substitution ρ such that

$$S\rho \subseteq S' \subseteq T\theta\,,$$

but since Σ does not contain the symbols $b_1, \ldots, b_m$, there is a substitution σ such that

$$S\sigma \subseteq T, \quad \text{and} \quad \sigma = \rho\theta^{-1}\,.$$

Thus, S subsumes T. □

Before proving Theorem 4.18, we show a property of SOL-S deductions in a special case in which a stronger result can be obtained.

If a sequence, $\mathbf{D}$, of structured clauses, $D_i, \ldots, D_n$, is a subsequence of an SOL-deduction (or SOL-S deduction), $D_0, \ldots, D_{i-1}, D_i, \ldots, D_n$, of S from $\Sigma + C$ and $\mathcal{P}$, we say $\mathbf{D}$ is an *SOL-deduction* (or *SOL-S deduction*) *of S from $\Sigma + D_i$ and $\mathcal{P}$*.

Lemma 4.30 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which C is used only as the top clause and the **Reduce** rule is never applied to any structured clause, then there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$.

Proof: Let $\mathbf{D} = D_0, \ldots, D_n$ be an SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$. Let l_i be the selected literal of $\vec{Q}_i$, where $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$ $(0 \leq i \leq n-1)$. Assume that **Reduce** is not used in the deduction. Then, we can ignore every framed literal for each $\vec{Q}_i$.

Let us suppose that there is a structured clause D_j in $\mathbf{D}$ such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$, but that **Resolve** is applied upon l_j in $\vec{Q_j}$ with a clause $B_j \in \Sigma$. Note that such a B_j is selected only from Σ by the supposition that C is never used as a side clause. We now call such D_j a *resister*. We prove the lemma by induction on the number m $(0 \leq m \leq n)$ of resisters in $\mathbf{D}$.

Firstly, we consider the case where $m = 0$. In this case, **Skip** is applied upon every l_j $(0 \leq j \leq n-1)$ such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$. Then, T is actually SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$, and of course $\Sigma \cup \{T\} \models T$ holds. Thus, the lemma holds for the base case.

Next, assume that the lemma holds for cases where there are at most $m-1$ $(m \geq 1)$ resisters in SOL-deductions. We consider the case where there are m resisters in $\mathbf{D}$. Let $D_k = \langle P_k, \vec{Q_k} \rangle$ $(0 \leq k \leq n-1)$ be the last resister in $\mathbf{D}$, namely k is the biggest number in such m resisters. Then $D_{k+1} = \langle P_{k+1}, \vec{Q_{k+1}} \rangle$ is obtained by applying **Resolve** to D_k with the clause B_k from Σ, and satisfies

$$\begin{array}{rcl} P_{k+1} & = & P_k\theta\,, \quad \text{and} \\ Q_{k+1} & = & (B_k\theta - \{\neg l_k\theta\}) \cup (Q_k\theta - \{l_k\theta\})\,, \end{array}$$

where θ is a substitution.

Obviously, T is SOL-deduced from $\Sigma + D_{k+1}$ and $\mathcal{P}$. Now, let U be a clause SOL-S deduced from $\Sigma + (B_k\theta - \{\neg l_k\theta\})$ and $\mathcal{P}$, V a clause SOL-S deduced from $\Sigma + (Q_k\theta - \{l_k\theta\})$ and $\mathcal{P}$. Since no more **Resolve** is applied upon any subsequent selected literal l_j $(j > k)$ such that $P_j \cup \{l_j\}$ belongs to $\mathcal{P}$, we can choose such U and V to satisfy

$$T = (P_k \cup U \cup V)\rho\,,$$

for some substitution ρ.

Now suppose that instead of applying **Resolve**, **Skip & Cut** is applied to D_k, deducing $D'_{k+1} = \langle P'_{k+1}, \vec{Q'_{k+1}} \rangle$, where

$$\begin{array}{rcl} P'_{k+1} & = & P_k \cup \{l_k\}\,, \quad \text{and} \\ Q'_{k+1} & = & Q_k - \{l_k\}\,. \end{array}$$

Then, $(P_k \cup \{l_k\} \cup V)\sigma$ for some substitution σ is SOL-S deduced from $\Sigma + D'_{k+1}$ and $\mathcal{P}$, and thus is SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$. Since $\Sigma \cup \{l_k\} \models B_k - \{\neg l_k\}$, $\Sigma \cup \{l_k\} \models U$ holds, and therefore it holds that

$$\Sigma \cup \{(P_k \cup \{l_k\} \cup V)\} \models T\,.$$

Now, since $S' = (P_k \cup \{l_k\} \cup V)$ is SOL-deduced from $\Sigma + C$ and $\mathcal{P}$ involving $m-1$ resisters, there is an SOL-S deduction of S from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \{S\} \models S' ,$$

by the induction hypothesis. Hence, $\Sigma \cup \{S\} \models T$. This completes the proof. □

We now take the **Reduce** rule into account. The result of Lemma 4.30 still holds even if factoring is incorporated. However, if in the SOL-deduction of T ancestry occurs upon a selected literal against a framed literal that had been resolved upon before a resister appeared, then no single SOL-S deduced clause S can satisfy the relation that $\Sigma \cup \{S\} \models T$. In this case, another SOL-S deduced clause S' is required to imply T, namely $\Sigma \cup \{S,\ S'\} \models T$ (see Example 4.20).

Theorem 4.18 If a clause T is derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which C is used only as the top clause, then there is a set δ of clauses each of which is derived by an SOL-S deduction from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models T$.

Proof: Let $\mathbf{D} = D_0, \ldots, D_n$ be an SOL-deduction of T from $\Sigma + C$ and $\mathcal{P}$. Let l_i be the selected literal of $\vec{Q}_i$, where $D_i = \langle\, P_i,\ \vec{Q}_i \,\rangle$ $(0 \leq i \leq n-1)$. We prove the theorem by induction on the number m $(0 \leq m \leq n)$ of **Reduce** operations in $\mathbf{D}$.

If **Reduce** is not applied upon any l_j $(0 \leq j \leq n-1)$, i.e., $m = 0$, then by Lemma 4.30 there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$. Thus the theorem holds for the base case.

Assume that the theorem holds for cases where **Reduce** operations occur $m-1$ or fewer times in SOL-deductions. We consider the case where there are m selected literals upon which **Reduce** is applied in $\mathbf{D}$. Let D_k be the last structured clause to which **Reduce** is applied in $\mathbf{D}$, namely k is the biggest number in such structured clauses. In this case, $D_{k+1} = \langle\, P_{k+1},\ \vec{Q_{k+1}} \,\rangle$ satisfies

$$\begin{aligned} P_{k+1} &= P_k\theta\,, \quad \text{and} \\ Q_{k+1} &= Q_k\theta - \{l_k\theta\}\,, \end{aligned}$$

where θ is a substitution. Note that T is SOL-deduced from $\Sigma + D_{k+1}$ and $\mathcal{P}$ without involving further application of **Reduce**.

1. Suppose that factoring (Rule 6(a)iiiA) is applied to D_k. In this case, since $\{P_k \cup Q_k\} \models P_{k+1} \cup Q_{k+1}$ holds, it is easy to verify that there is an SOL-deduction of S' from $\Sigma + D_k$ and $\mathcal{P}$ without involving any application of **Reduce** such that $\{S'\} \models T$. Since D_k is a structured clause in the SOL-deduction $\mathbf{D}$ of T, S' can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which $m - 1$ **Reduce** operations are applied. By the induction hypothesis, there is a set δ of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \delta \models S'$. Thus, $\Sigma \cup \delta \models T$.
2. Notice that in the above proof for applications of factoring, if all m **Reduce** operations are factoring, it holds that there is an SOL-S deduction of a clause S from $\Sigma + C$ and $\mathcal{P}$ such that $\Sigma \cup \{S\} \models T$. Hence, Lemma 4.30 holds even if factoring is taken into account. Thus, from now on, we can focus on induction on the number of ancestry (Rule 6(a)iiiB) operations in $\mathbf{D}$.
3. Assume that the theorem holds for cases where not more than $m-1$ ancestry operations are applied in SOL-deductions. Suppose that m ancestry operations occurs in $\mathbf{D}$. D_k is now the mth structured clause to which ancestry is applied. In this case, there is a structured clause $D_j = \langle P_j, \vec{Q_j} \rangle$ $(j < k)$ in $\mathbf{D}$ such that

$$\{P_j \cup Q_j, P_k \cup Q_k\} \models P_{k+1} \cup Q_{k+1}$$

(see the proof of Lemma 4.21). Now, apply **Resolve** to D_k against the clause $P_j \cup Q_j$, then apply **Skip** and factoring appropriately to the result. Then, we get an SOL-deduction of S_1 from $\Sigma \cup \{P_j \cup Q_j\} + D_k$ and $\mathcal{P}$ without involving any application of ancestry, such that

$$\{P_j \cup Q_j, S_1\} \models T\,.$$

Since D_k is a structured clause in the SOL-deduction $\mathbf{D}$ of T from $\Sigma + C$ and $\mathcal{P}$, S_1 can be derived by an SOL-deduction from $\Sigma \cup \{P_j \cup Q_j\} + C$ and $\mathcal{P}$. But since $\Sigma \cup \{C\} \models P_j \cup Q_j$ by Lemma 4.21, S_1 can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which $m - 1$ ancestry operations are applied. By induction hypothesis, there is a set δ_1 of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \delta_1 \models S_1\,.$$

In a similar way, apply **Resolve** to D_j against the clause S_1, then apply **Skip** and factoring appropriately to the result. Then, we get

an SOL-deduction of S_2 from $\Sigma \cup \{S_1\} + D_j$ and $\mathcal{P}$ without involving any application of ancestry, such that

$$\{S_1, S_2\} \models T\,.$$

Since D_j is a structured clause in the SOL-deduction $\mathbf{D}$ of T from $\Sigma + C$ and $\mathcal{P}$, S_2 can be derived by an SOL-deduction from $\Sigma \cup \{S_1\} + C$ and $\mathcal{P}$. But since S_1 is SOL-deduced from $\Sigma + C$ and $\mathcal{P}$, it holds that $\Sigma \cup \{C\} \models S_1$ and thus S_2 can be derived by an SOL-deduction from $\Sigma + C$ and $\mathcal{P}$ in which fewer than m ancestry operations are applied. Again, by the induction hypothesis, there is a set δ_2 of clauses SOL-S deduced from $\Sigma + C$ and $\mathcal{P}$ such that

$$\Sigma \cup \delta_2 \models S_2\,.$$

And therefore,

$$\Sigma \cup \delta_1 \cup \delta_2 \models T\,.$$

By letting $\delta = \delta_1 \cup \delta_2$, it holds that $\Sigma \cup \delta \models T$. This completes the proof of the theorem.

□

Chapter 5

Theorem Proving for Circumscription

In this chapter, as an application of consequence-finding procedures given in the last chapter, we investigate algorithms to answer queries in first-order logic databases augmented with the circumscription axiom. In the first half of this chapter, two recent proposals for such an algorithm are examined, and their relative merits are compared. We expand on the theoretical results that lead to such algorithms, make a detailed analysis of their computational properties and propose a more general procedure that leads to computational savings.

In the latter half of this chapter, we address the problem of answering queries in circumscription and related nonmonotonic formalisms. In a way analogous to query answering techniques in the classical predicate logic, the process extracts information from a proof of the query. Circumscriptive theorem provers consist of two processes, generating explanations for the theorem to be proved and showing that these explanations cannot be refuted. In general, many explanations compete in supporting the theorem. We show that queries can be answered by finding certain combinations of explanations, and present some results on how to search the space of explanations, while incorporating significant pruning on this space. The results are relevant to other nonmonotonic formalisms having explanation-based proof procedures.

5.1 Introduction

Circumscription [McCarthy, 1980; Lifschitz, 1985] is one of the most powerful and well-developed formalizations of nonmonotonic reasoning. Although its formal properties are well investigated, there have been few attempts at effective query answering procedures or implementations for circumscriptive theories.

Recently, Przymusinski [1989] and Ginsberg [1989] have published algorithms to compute circumscription for ground theories. We are concerned with these procedures as they are simple extensions of theorem provers for first-order logic. While these procedures might be good, they suffer from a number of problems.

Firstly, because of different concerns and formalisms of [Przymusinski, 1989; Ginsberg, 1989], not much is known about the algorithms' relative advantages and disadvantages and about their limitations for nonground theories.

Secondly, there is an important drawback in these procedures; namely the algorithms give only "Yes/No" answers, that is, if the query contains variables, the values for it are not returned. This is clearly a serious drawback if we wish to use circumscription as part of a knowledge representation system.

Therefore, the goal of this chapter is twofold:

1. We explore the connections between two algorithms [Przymusinski, 1989; Ginsberg, 1989], showing that:

   (a) The theoretical results obtained in each of these papers are similar, and both can be re-expressed in a simple, general framework.

   (b) The algorithms presented have different computational properties; we provide a detailed comparison between them.

   Then, we show how the efficiency of both algorithms can be improved.

2. We propose a solution to the answer extraction problem, and show how this solution can help improve efficiency.

In the first half of this chapter, Sections 5.2, 5.3, and 5.4 address the first question in the above two points. In the latter half of this chapter, Sections 5.5 and 5.6 illustrate two important problems that arise in Przymusinski's and Ginsberg's approaches through examples, and Section 5.7 provides the main results on extracting answers from a proof.

5.2 Background

We briefly recall a basic property of circumscription, on which the algorithms are based. The reader is also referred to Section 2.2.2 for introduction of circumscription. While circumscription can be expressed by the second-order axiom, we use the model theory of circumscription [McCarthy, 1980; Lifschitz, 1985] directly in the following discussion. The predicate symbols of a first-order theory T are divided into three tuples called the *circumscription policy*: $\mathbf{P}$, *minimized* predicates; $\mathbf{Z}$, *variables*; and

Q, *fixed*. We will use each of **P**, **Q** and **Z** as a set of predicates as well as a tuple of them in this chapter. Using this information, some models of T are defined as minimal with respect to the sets **P** and **Z** (see Definition 2.7). Let $CIRC(T;\mathbf{P};\mathbf{Z})$ be the *circumscription of* **P** *in* T *with* **Z**. Then, by Theorem 2.8, for any formula F, $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if every $(\mathbf{P},\mathbf{Z})$-*minimal model* M of T satisfies F.

General circumscription is highly uncomputable [Schlipf, 1986], and existing theorem proving results apply to restricted circumscriptive theories. From now on, T is a first-order theory without equality, consisting of finitely many clauses. We assume that T contains equality axioms and that the *unique-names assumption* (UNA) holds in T, that is, different ground terms denote different elements of the domain. We also assume that the *domain-closure assumption* (DCA) is satisfied, that is, the only objects in the domain are the ones that can be named using the object and function constants in the language, since this is necessary to guarantee the soundness of the query answering procedure described in this chapter. Queries to be answered are restricted to existentially quantified formulas (note that this includes ground formulas). Note that these assumptions are used in [Gelfond *et al.*, 1989] (see Section 2.3.2) and are sufficient to illustrate the comparison between the theoretical results of [Przymusinski, 1989; Ginsberg, 1989] as they consider only the *ground* case, in which T is a set of finitely many ground clauses (so that DCA is not necessary) and queries are ground clauses, which is a special case of ours. Additional results concerning these assumptions for circumscription in restricted classes of theories can be found in [Gelfond *et al.*, 1990].

5.2.1 Comparing the Theorems

Given a theory T with the circumscription policy **P** and **Z**, we want to know whether a formula F is a theorem of $CIRC(T;\mathbf{P};\mathbf{Z})$. Historically, Bossu and Siegel [1985] first computed circumscription using theorem proving techniques when both the fixed predicates **Q** and variables **Z** are empty. More recently, Przymusinski [1989] and then Ginsberg [1989] have developed such algorithms. In contrast to the reduction of the circumscription axiom to first-order theories proposed by Lifschitz [1985] and others, these theorem provers allow us the computation of the circumscription of the theories that cannot be handled by those reduction methods. Furthermore, the theorem proving techniques can be applied more easily to these results while the resultant theories obtained by a reduction method is generally very complex first-order formulas.

However, both algorithms of [Przymusinski, 1989; Ginsberg, 1989] are relatively complex: the former is based on MILO-resolution, a variant of ordered linear resolution; the latter uses a backward-chaining ATMS. Because of their different concerns

and formalisms in addition to their different implementation styles, it is not clear what kind of theories may lead to such algorithms. We will make a detailed comparison of the formal theories that lead to their algorithms by relating them to a logical framework of abduction, and explain their intuitive meaning.

We will use the notion of (new) characteristic clauses which was introduced in Chapter 3. By means of these notions, we will give a simple formulation and interpretation for computing circumscription in the following subsections. Throughout this chapter, however, we denote by $Carc(T)$ the *characteristic clauses* of a theory T (with respect to a production field $\mathcal{P}$). Note that the correct notation of Definition 3.4 is $Carc(T, \mathcal{P})$ since it depends on the production field $\mathcal{P}$. As there will be no confusion with $\mathcal{P}$, however, we simply write $Carc(T)$. The same treatment is applied to the *new characteristic clauses* of a formula F with respect to T and $\mathcal{P}$ (Definition 3.6), which is denoted $Newcarc(T, F)$ by omitting $\mathcal{P}$. In this chapter, we also ignore all the valid clauses in $Carc(T)$ as they are of no use in our purposes.

Recall that if $\mathbf{R}$ is a set of predicate symbols, $\mathbf{R}^+$ (respectively $\mathbf{R}^-$) means the positive (respectively negative) literals with predicates from $\mathbf{R}$ in the language. Now, we denote by $\mathbf{R}^\pm$ the set of all occurrences of predicates from $\mathbf{R}$, i.e., $\mathbf{R}^\pm = \mathbf{R}^+ \cup \mathbf{R}^-$.

5.2.2 Przymusinski's Results

Przymusinski's [1989] algorithm is based on the two theorems (Theorems 5.1 and 5.2 below) developed by Gelfond *et al.* [1989], which provide the connection between the computation of circumscription and the use of extensions of the resolution-based query evaluation procedures. The first theorem characterizes the computation for a query which does not contain literals from the variables $\mathbf{Z}$.

Theorem 5.1 [Przymusinski, 1989, Theorem 2.5]
If a formula F does not contain literals from $\mathbf{Z}$, then $CIRC(T; \mathbf{P}; \mathbf{Z}) \models F$ if and only if there is no clause E such that (i) E does not contain literals in $\mathbf{Z}^\pm \cup \mathbf{P}^-$, and (ii) $T \models \neg F \vee E$ but $T \not\models E$.

Now let us rewrite this theorem using the new characteristic clauses. Condition (i) means that E belongs to the production field

$$\mathcal{P} = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle\,.$$

$T \models \neg F \vee E$ can be written as $T \cup \{F\} \models E$. So, we are looking for a clause E belonging to the production field, implied by $T \cup \{F\}$ but not by T alone. This means that $E \in Th_{\mathcal{P}}(T \cup \{F\}) - Th(T)$. Now, for a set of clauses Σ, $\Sigma = \emptyset$ if and only if $\mu[\Sigma] = \emptyset$. Therefore, it is enough to check whether $Newcarc(T, F)$ is empty or not (by Definition 3.6). That is,

Theorem 5.1 (new version) Let F be a formula not containing literals from $\mathbf{Z}$. Let $\mathcal{P}$ be $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle$. Then,

$$CIRC(T;\mathbf{P};\mathbf{Z}) \models F \quad \text{if and only if} \quad Newcarc(T,F) = \emptyset.$$

This formulation helps to understand the intuition underlying the above theorem. We want to know if a query F not involving literals from $\mathbf{Z}$ is true or not in the $(\mathbf{P},\mathbf{Z})$-minimal models of a theory T. Remember that every $(\mathbf{P},\mathbf{Z})$-minimal model of T is defined on the interpretations of T by considering differences of the extensions of $\mathbf{P}$ and equality of the extensions of $\mathbf{Q}$, but by ignoring differences of the extensions of $\mathbf{Z}$ (Definition 2.7). Now, the minimal models of $Carc(T)$ (where $\mathcal{P} = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle$) are equivalent to the partial models of T that are the same as the minimal models of T but without the relations for the predicates belonging to $\mathbf{Z}$. Therefore, the characteristic clauses of T are representative of those minimal models, in the sense that if adding F to T produces a change (new one) in $Carc(T)$ then the addition of F has produced a change in the minimal models of T as well. The existence of a new characteristic clause of F means that F has altered the minimal models: thus if $Newcarc(T,F)$ is empty, the addition of F has no effect on the minimal models and the circumscriptive theory entails it.

For formulas containing predicates from the variables $\mathbf{Z}$, the following theorem originally developed by Gelfond *et al.* [1989] holds.

Theorem 5.2 [Przymusinski, 1989, Theorem 2.6]
Let F be any formula. $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if either $T \models F$ or there is a formula G such that (i) G does not contain literals in $\mathbf{Z}^\pm \cup \mathbf{P}^-$, (ii) $T \models F \vee G$, and (iii) $CIRC(T;\mathbf{P};\mathbf{Z}) \models \neg G$.

Now, $T \models F$ means that $T \cup \{\neg F\}$ is unsatisfiable; note that in this case, $Newcarc(T,\neg F)$ contains only $\Box$ (the empty clause). Condition (i) again means that G belongs to $\mathcal{P} = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle$; condition (ii) can be written as $T \cup \{\neg F\} \models G$; and condition (iii) is equivalent to $Newcarc(T,\neg G) = \emptyset$ by Theorem 5.1. In this case, the condition $T \not\models G$ is missing in Theorem 5.2; if $T \models G$, however, then $Newcarc(T,\neg G) = \{\Box\} \neq \emptyset$ holds for satisfiable T. Therefore, condition (ii) together with (iii) further implies that G is in the form of a conjunction of some clauses from $Newcarc(T,\neg F)$. [1] Finally, in condition (iii), if G is $\Box$, then $\neg G$ is the formula *true* and adding it to T produces no new theorem: $Newcarc(T, true) = \emptyset$. We can now write:

[1] In practice, the minimality condition involved by the μ operation is not crucial. See Section 5.3.1.

Theorem 5.2 (new version) Let F be any formula, and $\mathcal{P} = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle$. $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if there is a conjunction G of some clauses from $Newcarc(T, \neg F)$ such that $Newcarc(T, \neg G) = \emptyset$.

While this formulation seems simpler than the original one, it still does not provide much insight. We will see it more clearly in Section 5.2.3, relating it with abduction.

Example 5.3 Let us verify Theorem 5.2 by reviewing the "birds-fly" example given in Example 2.6. The theory T can be written in the following clausal form:

$$\begin{aligned} &\neg Bird(x) \vee Ab(x) \vee Flies(x)\,, \\ &\neg Penguin(x) \vee Bird(x)\,, \\ &\neg Penguin(x) \vee \neg Flies(x)\,. \end{aligned}$$

In this example, the predicates were divided into $\mathbf{P} = \{Ab\}$, $\mathbf{Z} = \{Flies\}$, and $\mathbf{Q} = \{Bird, Penguin\}$. Let us fix $\mathcal{P}$ to be $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle$, that is, positive occurrences of Ab, or any occurrence of $Bird$ and $Penguin$:

$$\mathcal{P} = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^\pm \,\rangle = \langle\, \{Ab\}^+ \cup \{Bird, Penguin\}^\pm \,\rangle$$

Then, the characteristic clauses of T (without $Taut(\mathbf{Q}^\pm)$) are given as:

$$Carc(T) = \{\ \begin{array}{l} \neg Penguin(x) \vee Bird(x)\,, \\ \neg Penguin(x) \vee Ab(x) \end{array} \ \}\,.$$

Let us consider the first query, "Can any bird fly?": [2]

$$F_1 = (\, Bird(Tweety) \supset Flies(Tweety) \,)\,.$$

Adding $\neg F_1 = (\, Bird(Tweety) \wedge \neg Flies(Tweety) \,)$ to T gives

$$Newcarc(T, \neg F_1) = \{\, Bird(Tweety),\ Ab(Tweety) \,\}.$$

Since adding the negation of the conjunction of these two unit clauses,

$$\neg Bird(Tweety) \vee \neg Ab(Tweety)\,,$$

to T gives a new characteristic clause,

$$\neg Bird(Tweety) \vee \neg Penguin(Tweety)\,,$$

[2] Since $Tweety$ does not appear anywhere in T, it can be regarded as a Skolem constant of $\neg F_1$ as if the query F_1 is obtained from the formula $\forall\, x\,(\, Bird(x) \supset Flies(x) \,)$. Similarly, $CIRC(T;\mathbf{P};\mathbf{Z}) \models F_2$ implies that $\forall\, x\,(\, Bird(x) \wedge \neg Penguin(x) \supset Flies(x) \,)$ is a theorem of the circumscriptive theory.

it holds that

$$CIRC(T;\mathbf{P};\mathbf{Z}) \not\models F_1 .$$

Now consider the second query, "Can any bird which is not a penguin fly?":

$$F_2 = (\, Bird(Sam) \wedge \neg Penguin(Sam) \supset Flies(Sam)\,) .$$

Adding the negation of the query to T gives

$$Newcarc(T, \neg F_2) = \{\, Bird(Sam),\ \neg Penguin(Sam),\ Ab(Sam)\,\}.$$

Adding the negation of the conjunction of these three clauses to T produces no new characteristic clauses, showing that

$$CIRC(T;\mathbf{P};\mathbf{Z}) \models F_2 .$$

5.2.3 Ginsberg's Results

Ginsberg [1989] presents another algorithm for computing circumscription. The algorithm, however, works only in the case where $\mathbf{Q}$, the set of fixed predicates, is empty. We will transform Ginsberg's definitions and results to ours.

Definition 5.4 [Ginsberg, 1989, Definition 3.1]
Let H and T be two sets of formulas. A formula G is *dnf* with respect to H if it is written as a disjunction of conjunctions of some elements in H. A formula F is *confirmed by* G (for T and H) if the following conditions hold:

1. $T \cup \{G\}$ is satisfiable,
2. $T \cup \{G\} \models F$, and
3. G is dnf with respect to H.

Comparing Definition 5.4 with Definition 3.10, we see that F is confirmed by G for T and H if G is a disjunction of *explanations of* F *from the abductive framework* (T, H). By Proposition 3.11, $\neg G$ is a conjunction of some clauses each of which belongs to the production field $\langle\, \overline{H}\,\rangle$. In other words,

Definition 5.4 (new version) Let $\mathcal{P} = \langle\, \overline{H}\,\rangle$. F is confirmed by G if $\neg G$ is a conjunction of some clauses from $Newcarc(T, \neg F)$. Moreover, F is *unconfirmed*, if no G confirms F: $Newcarc(T, \neg F) = \emptyset$.

Next is the main result:

Proposition 5.5 [Ginsberg, 1989, Proposition 3.2]
Let H be $\mathbf{P}^-$. $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if there is some G confirming F such that $\neg G$ is unconfirmed.

We can rewrite it as:

Proposition 5.5 (new version) Let $\mathcal{P} = \langle\, \overline{H} \,\rangle = \langle\, \mathbf{P}^+ \,\rangle$. $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if there is a conjunction G of some clauses from $Newcarc(T, \neg F)$ such that $Newcarc(T, \neg G) = \emptyset$.

Ginsberg briefly mentions connections with Przymusinski's work and the possibility of relaxing the assumption of all non-minimized predicates being variable. Our above results show that:

1. This last proposition is exactly Theorem 5.2.
2. All results can thus be extended to the case $\mathbf{Q} \neq \emptyset$ (that is, not all predicates are allowed to vary) just by setting $H = \mathbf{P}^- \cup \mathbf{Q}^{\pm}$, i.e., $\mathcal{P} = \langle\, \overline{H} \,\rangle = \langle\, \mathbf{P}^+ \cup \mathbf{Q}^{\pm} \,\rangle$.

The intuition behind Theorem 5.2 and Proposition 5.5 is the following. From the viewpoint of abductive reasoning, those theorems say that $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ *if and only if there is a disjunction* G *of explanations from* (T, H) *such that there exists no explanation of* $\neg G$ *from* (T, H).[3] For answering queries in circumscription, the hypotheses H must be carefully chosen in the direction of $(\mathbf{P}, \mathbf{Z})$-minimization:

$$H = \mathbf{P}^- \cup \mathbf{Q}^{\pm}\,,$$

namely, for minimized predicates $\mathbf{P}$, $\mathbf{P}^-$ should be hypothesized, and for fixed predicates $\mathbf{Q}$, $\mathbf{Q}^{\pm}$ should be taken into account so that the definitions of $\mathbf{Q}$ might not be altered. Now the existence of an explanation of F from (T, H) guarantees that F holds in at least one *extension* of (T, H) by Proposition 3.13. Clearly, if some disjunction G of explanations of F holds in all extensions, then F also holds in all extensions. Since this G is constructed over H and thus does not contain literals from $\mathbf{Z}$, we see that G holds in all extensions of (T, H) if and only if $Newcarc(T, G) = \emptyset$ with respect to $\mathcal{P} = \langle\, \overline{H} \,\rangle$ (by Theorem 5.1) if and only if there is no explanation of $\neg G$ from (T, H) (by Proposition 3.11). Hence, we get Theorem 2.14 in Chapter 2.

[3] Etherington [1988] has shown the equivalence of membership in all extensions and circumscriptive entailment for propositional theories without fixed predicates. Also, Poole [1989] introduces the similar condition for a formula to hold in all extensions of (T, H), and Lin and Goebel [1989] independently derive the equivalent theorem from the result by [Gelfond *et al.*, 1989] within the Theorist framework [Poole *et al.*, 1987]. See Section 2.3.2 in Chapter 2.

5.3 Query Answering Procedures

In the last section we showed that both Przymusinski's and Ginsberg's algorithms were based on the same theoretical results. This section concerns the computational efficiency of the algorithms.

Przymusinski [1989] defines *MILO-resolution*, a variant of ordered linear (OL) resolution [Chang and Lee, 1973]. Given a clause C, MILO-resolution is used to deduce a set of minimal clauses belonging to $Th_{\mathcal{P}}(T \cup \{C\})$, called the *MILO-derivative of* $T + C$, with top clause C and the theory T. The algorithm needs to check the non-deducibility from T of each clause in the MILO-derivative, in order to determine the new characteristic clauses. On the other hand, Ginsberg's [1989] circumscriptive theorem prover uses a "backward-chaining ATMS" [Reiter and de Kleer, 1987] to compute minimal explanations of formulas. This backward chaining procedure also uses a classical theorem prover.

While the structure of the proofs are similar, each algorithm has a different concern and extends a resolution procedure in a different way. Remember that we should produce clauses (i) in the production field, and (ii) the "new" and "minimal" of these, that is, neither implied by the original theory nor by another produced clause. MILO-resolution provides the ability to restrict the resolution to some literals by which the algorithm directly focuses on producing the clauses relevant to answer the query, that is, those in the production field $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^{\pm} \,\rangle$. Przymusinski's concern is thus efficiency regarding the first of the above two points. Ginsberg uses a classical theorem prover; this means that no information concerning the production field is used during the proof. His algorithm has however another concern, that of the minimality of the produced formulas. For this, he uses a structure called a "bilattice" based on his previous work on multivalued logic [Ginsberg, 1988]. The role of this bilattice is to record inferences, in order to avoid making them more than once. He is thus concerned with the second of the above two points.

The next two subsections expand on these ideas. The discussion is based on each resolution procedure to compute $Newcarc(T, C)$, given a theory T, a clause C and a production field $\mathcal{P}$.

5.3.1 What Needs to be Computed

From the results presented above, it appears that, in order to answer a query F in a circumscriptive theory $CIRC(T; \mathbf{P}; \mathbf{Z})$, an algorithm should first compute the minimal explanations of the formula F from $(T, \mathbf{P}^- \cup \mathbf{Q}^{\pm})$, or equivalently their negations, $Newcarc(T, \neg F)$, with the production field set to $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^{\pm} \,\rangle$. Ginsberg's

theorem prover works exactly along this line of computation.[4]

However, there is a set smaller than $Newcarc(T, F)$ that can be used to answer such a query. In a way analogous to the discussion given in Section 4.3.2, the set of clauses $\mathcal{S}$ produced by using deductions with top clause C, the theory T and the production field $\mathcal{P}$, can be divided into two sets, $\mathcal{S}_1$ and $\mathcal{S}_2$, such that

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \text{ and } T \cup \mathcal{S}_1 \models \mathcal{S}_2 .$$

Adding $\mathcal{S}_2$ to $\mathcal{S}_1$ does not change the models of the produced clause, so only $\mathcal{S}_1$ needs to be computed. We call such a set $\mathcal{S}_1$ a *precursor of* $\mathcal{S}$. Note that a clause in a precursor may not belong to $Newcarc(T, C)$. MILO-resolution actually computes such a precursor as the MILO-derivative of $T + C$, because it restricts the resolution to literals belonging to $\mathbf{Z}^{\pm} \cup \mathbf{P}^{-}$. In other words, when the first literal of the center clause belongs to $\mathcal{P} = \langle\, \mathbf{P}^{+} \cup \mathbf{Q}^{\pm} \,\rangle$, it is *skipped*. If it were resolved upon with a clause from the theory, the resulting leave obtained by chaining the inference would be implied by the one obtained with the skip operation (see Theorem 5.8). Therefore, we can conclude that MILO-resolution performs a kind of *SOL-S deductions* given in Section 4.3.2, and that the MILO-derivative corresponds to the *S-production*. This is best understood with an example.

Example 5.6 Consider the circumscription of $\mathbf{P}$ in T with $\mathbf{Z}$, where the theory is

$$T = \{\, P_1 \vee \neg P_2, \quad P_2 \vee \neg P_3, \quad P_3 \vee G \,\} ,$$

the minimized predicates are $\mathbf{P} = \{P_1, P_2, P_3\}$, and the variables are $\mathbf{Z} = \{G\}$. The production field is $\mathcal{P} = \langle\, P^{+} \,\rangle = \langle\, \{P_1, P_2, P_3\} \,\rangle$. The query is G.

Now, by adding $\neg G$ to T, an SOL-S deduction provides P_3, which is the MILO-derivative of $T + \neg G$ too. Since this literal belongs to $\mathcal{P}$, the procedure skips it and stops. Then adding $\neg P_3$ to T generates no new characteristic clause. According to Theorem 5.2, G is a theorem of the circumscriptive theory.

If the procedure were to examine the remaining choice, resolving P_3 with the clause $P_2 \vee \neg P_3$, it would produce P_2, and a further step would produce P_1. This is exactly what SOL-resolution (without preferring Skip rule) and Ginsberg's prover may do, and the production from $T + \neg G$ and $\mathcal{P}$ can provide:

$$Newcarc(T, \neg G) = \{P_3,\, P_2,\, P_1\} .$$

The negation of the CNF formula is $\neg P_3 \vee \neg P_2 \vee \neg P_1$, which provides no new characteristic clauses with respect to T, verifying the result of the above computation. The SOL-S deduction did not need to generate two extra clauses because $\{P_3\}$ is a precursor of them:

$$T \cup \{P_3\} \models P_1 \wedge P_2 .$$

[4] This is in essence what [Lin and Goebel, 1989] does too.

There is another big difference between Przymusinski's and Ginsberg's provers concerning checking the consistency of hypotheses. Recall that, to apply Theorem 5.2 or Proposition 5.5, we need two steps; firstly computing a set of clauses belonging to $Newcarc(T, \neg F)$ for the query F, then checking whether a conjunction G of those clauses satisfies that $Newcarc(T, \neg G) = \emptyset$. Ginsberg's prover first computes the minimal explanations $\mathcal{E}$ of F from (T, H), then tries to compute the minimal explanations of $\neg \bigvee_{E_i \in \mathcal{E}} E_i$ from (T, H). On the contrary, Przymusinski's prover first computes the MILO-derivative $\mathcal{D}_1$ of $T + \neg F$, without checking the non-deducibility from T of each clause in $\mathcal{D}_1$, then tries to compute the MILO-derivative $\mathcal{D}_2$ of $T + \neg \bigvee_{A_j \in \mathcal{D}_1} A_j$, checking whether $T \not\models B$ for each clause B in $\mathcal{D}_2$ *one by one*. Clearly, in the second step, we need not compute all the minimal explanations for the negation of the disjunction; if it has at least one explanation then we can stop the computation immediately. For the first step, Przymusinski's prover may include some clauses belonging to $Th_{\mathcal{P}}(T)$ in $\mathcal{D}_1$, which are excluded from the produced clauses by Ginsberg's prover. [5] However, since it takes much computation for this consistency checking (non-decidable for the first-order case), it seems rather efficient even if these extra clauses are included in the second step. Therefore, the efficiency may depend on the knowledge base.

5.3.2 How it is Computed

Now suppose both algorithms have to compute the same set, that is, MILO-resolution computes all the new characteristic clauses by avoiding the skip of literals, or Ginsberg's theorem prover is restricted to computing a precursor. In that case, another advantage of using the information on the production field $\mathcal{P}$ during the deduction is that fewer clauses are generated.

For a very simple example, suppose that the center clause is $Z_1 \vee Q_1$, where Z_1 does not belong to $\mathcal{P}$, while Q_1 belongs to $\mathcal{P}$. If Z_1 cannot be resolved upon against clauses of the theory in such a way that the result of the deduction produces a clause belonging to $\mathcal{P}$, MILO-resolution will never try to resolve on the next literal Q_1. Conventional theorem provers will give no priority to Z_1 over Q_1 and thus will try all the resolutions on Q_1 as well. This example, although trivial, is representative of what will happen in many realistic situations.

We said above that a central concern of [Ginsberg, 1989] was to avoid computing the same clauses more than once. The role of the bilattice is to record information and use it to avoid redundant derivations by making subsumption tests. Avoiding the

[5] If there is a clause $A \in \mathcal{D}_1$ such that $T \models A$, then in the second step, the negation of the disjunction should contain $\neg A$ as its disjunct. Since its valuation is false, this does not affect the result of the query answering.

exploration of unnecessary portions of the search space, and in particular the non-production of subsumed clauses has been a central concern of automated theorem proving and is one of the motivations behind all the refinements of resolution [Loveland, 1978]. Many of these use the information of literals that have been resolved upon to avoid producing many of redundant clauses. For example, OL-resolution on which MILO-resolution uses *framed literals* (or, *A-literals*) to record the history of the deduction. Regarding other problems related to irredundancy and control, a thorough analysis of subsumption can be found in [Loveland, 1978, Chapter 4]. There is not enough information in [Ginsberg, 1988; Ginsberg, 1989] to determine whether the bilattice represents a better alternative.

5.4 Improving Efficiency

We show here how search space of MILO-resolution can be reduced. MILO-resolution is based on Chang and Lee's [1973] version of OL-resolution; this procedure is augmented with the ability to skip literals when they belong to the production field.

Now, there exist superior versions of linear resolution that can be augmented with skip operations; most notably, Model Elimination [Loveland, 1978] and SL-resolution [Kowalski and Kuhner, 1971]. Basically, the Model Elimination procedure introduced the restriction that, without loss of completeness, it can *avoid resolving the center clause $\vec{C}$ against a clause that contains a literal k appearing in $\vec{C}$ as the framed literal $\boxed{k}$ at the right of the literal resolved upon*, as such a resolution would produce only clauses subsumed by some previous center clause. Notice that SOL-resolution given in the last chapter contains this kind of restriction in Rule 4 of Definition 4.1 (see Remark (2) of Definition 4.1). Clearly, this has two advantages: it restricts the search space, and avoids many of the subsumption tests in Step (iii) of MILO-resolution [Przymusinski, 1989, Definition 3.1].

Example 5.7 Suppose a clause,

$$F = P \vee R\,,$$

resolves with a clause,

$$\neg P \vee Q\,,$$

in T, giving

$$Q \vee \boxed{P} \vee R\,.$$

Now suppose there is a clause,

$$\neg Q \vee P\,,$$

in T. The above restriction tells us that this clause may safely be disregarded in the deduction, as it contains P, a literal appearing as the framed literal $\boxed{P}$ in the center clause. In effect, it would give the clause,

$$P \vee \boxed{Q} \vee \boxed{P} \vee R\,,$$

which is subsumed by a previous center clause, F.

Additional improvements in efficiency were introduced by [Shostak, 1976] and [Bibel, 1982]. Shostak [1976] shows that we can record still more information on center clauses. When the first literal of the center clause is framed, instead of deleting it from the clause, Shostak's procedure complements it and keeps it in a different position called the *C-point* in the clause. Such *C-literals* can still be used later in the reduction to reduce the search space and do ancestor resolution as with the ordinary framed literals.

Example 5.7 (continued) If now the above clause,

$$Q \vee \boxed{P} \vee R\,,$$

is resolved with,

$$\neg Q\,,$$

the result will be

$$R \vee (\neg Q) \vee (\neg P)\,,$$

where, the notation $(\neg l)$ is used for the truncated framed literal $\boxed{l}$ moved to the C-point; thus the information that $\neg P$ and $\neg Q$ are proved is kept.

5.4.1 SOL-S Resolution

We thus propose the following procedure schema. Given a set of clauses T, a clause C, and a production field $\mathcal{P}$, a deduction of a clause S from $T + C$ (the background theory $T \cup \{C\}$ with top clause C) and $\mathcal{P}$ consists of a sequence of structured clauses, $C_0, C_1, \ldots, C_n$, such that:

1. $C_0 = \langle\, \Box,\, C\,\rangle$,
2. $C_n = \langle\, S,\, \Box\,\rangle$, and
3. $C_{i+1} = \langle\, S_{i+1},\, R_{i+1}\,\rangle$ is obtained from $C_i = \langle\, S_i,\, R_i\,\rangle$ by applying the following operations. We assume that R_i is ordered and l is the selected literal of R_i.

(a) **(Skip)** If l belongs to $\mathcal{P}$, then $S_{i+1} = S_i \vee l$ and R_{i+1} is obtained by removing l from R_i.

(b) Otherwise, $S_{i+1} = S_i$, and R_{i+1} is obtained by a linear resolution procedure where the center clause is R_i and the background theory is $T \cup \{C\}$.

In Rule 3(b), we can use any linear resolution procedure that is complete for refutation. By using this modified procedure, we can find a precursor without computing all of $Newcarc(T, C)$.

Theorem 5.8 (Variant of Theorem 4.18)
If a clause K belongs to $Newcarc(T, C)$, then there is a set δ of clauses, each of which is derived from $T + C$ and $\mathcal{P}$, such that $T \cup \delta \models K$. □

The query answering procedure for circumscriptive theories [Przymusinski, 1989, Algorithm 4.1] that calls MILO-resolution remains unchanged; the definition of the MILO-derivative of $T + C$ is just changed to be the output of the above procedure instead of the output of MILO-resolution.

In Chapter 4, we have called the above deduction an *SOL-S deduction* (SOL-deduction preferring Skip rule). SOL-S resolution is a variant of SOL-resolution [Inoue, 1991b] defined for first-order theories in the last chapter, which incorporates the restriction rule used in Example 5.7. The only difference between an SOL-S deduction and an SOL deduction is that while the rules 3(a) and 3(b) in the former are not alternative, i.e., for every selected literal l belonging to $\mathcal{P}$ only the **Skip** rule can be applied; but in the latter they are alternative choices, i.e., for l belonging to $\mathcal{P}$ either rule can be applied. While Theorem 4.3 (2) guarantees that SOL-resolution is complete for finding new characteristic clauses, Theorem 4.18 ensures that SOL-S resolution can find a precursor.

5.4.2 Example

We might use Shostak's GC procedure as a linear resolution procedure for Step 3(b) above, and get the following:

Example 5.9 (modified version of [Przymusinski, 1989, Example 3.5]) We apply the procedure to formulas with variables such that deductions with them remain to be ground. Let T contain the following formulas, with $\mathbf{P} = \{Learns, Senior\}$, and $\mathbf{Z} = \emptyset$.

$$\forall x\,(Senior(x) \supset Learns(x, Latin) \vee Learns(x, Greek))\,,$$
$$\forall x\,(Senior(x) \supset Learns(x, French))\,,$$
$$Senior(Ann)\,,$$
$$\forall x\,(Learns(x, Greek) \supset Senior(x))\,.$$

In clausal form and with the obvious abbreviations, the theory is

$$L(x, Lt) \vee L(x, Gr) \vee \neg S(x), \tag{1}$$
$$\neg S(x) \vee L(x, Fr), \tag{2}$$
$$S(A), \tag{3}$$
$$\neg L(x, Gr) \vee S(x). \tag{4}$$

The production field $\mathcal{P}$ is then $\langle\, \mathbf{P}^+ \,\rangle = \langle\, \{L, S\}^+ \,\rangle$. Consider the query

$$\neg L(A, Gr) \vee \neg L(A, Fr)\,.$$

The following is a deduction obtained by our procedure.

$C_0 = \langle\, \Box,\ \neg L(A, Gr) \vee \neg L(A, Fr) \,\rangle$ —Given.

$C_1 = \langle\, \Box,\ L(A, Lt) \vee \neg S(A) \vee \boxed{\neg L(A, Gr)} \vee \neg L(A, Fr) \,\rangle$
—Resolution with 1 (*).

$C_2 = \langle\, L(A, Lt),\ \neg S(A) \vee \boxed{\neg L(A, Gr)} \vee \neg L(A, Fr) \,\rangle$
—Skip the literal from $\mathcal{P}$ (**).

$C_3 = \langle\, L(A, Lt),\ \boxed{\neg S(A)} \vee \boxed{\neg L(A, Gr)} \vee \neg L(A, Fr) \,\rangle$
—Resolution with 3.

$C_4 = \langle\, L(A, Lt),\ \neg L(A, Fr) \vee \big(S(A)\big) \vee \big(L(A, Gr)\big) \,\rangle$
—Recording of solved literals.

$C_5 = \langle\, L(A, Lt),\ \neg S(A) \vee \boxed{\neg L(A, Fr)} \vee \big(S(A)\big) \vee \big(L(A, Gr)\big) \,\rangle$
—Resolution with 2.

$C_6 = \langle\, L(A, Lt),\ \boxed{\neg L(A, Fr)} \vee \big(S(A)\big) \vee \big(L(A, Gr)\big) \,\rangle$
—Reduction using the solved literal (***).

$C_7 = \langle\, L(A, Lt),\ \Box \,\rangle$ —Truncation.

Now according to Theorem 5.1, we can answer "No" to the above query, that is,

$$CIRC(T; \mathbf{P}; \mathbf{Z}) \not\models \neg L(A, Gr) \vee \neg L(A, Fr)\,,$$

because $L(A, Lt)$ is not implied by T.

Let us look at some advantages of this deduction over MILO-resolution and Ginsberg's theorem prover.

- At point (*), Ginsberg's prover, which lacks information on the production field, will resolve on $L(A, Lt)$, instead of skipping it, and explore the branches that are pruned by this skip operation.

- At point (**), MILO-resolution will behave as in the above deduction, but keeps an additional choice that results from resolving $\neg S(A)$ with clause 4. Our procedure avoids this because clause 4 contains the literal $\neg L(A, Gr)$ that appears framed in the center clause, thus indicating the remaining choice is unnecessary.

- At point (***), MILO-resolution would have lost information about $S(A)$, and thus makes a resolution against all clauses containing $\neg S(A)$. Reduction with the solved literal avoids this exploration.

5.4.3 Remarks

The improvements in the present section are mainly based on direct refinements of linear resolution procedures, and actually applicable to efficient computation of $Newcarc(T, \neg F)$ for a query F and $Newcarc(T, \neg G)$ for some G in Theorem 5.2.

Other techniques of theorem proving can be used to improve efficiency still more. For example, we can "compile" the theory, producing either its prime implicates (see Definition 3.14) or the sub-clauses implied by it (see Definition 3.21). In both cases, the resultant theory has the same models, and thus the same $(\mathbf{P}, \mathbf{Z})$-minimal models as the former (see Lemma 3.15 and Proposition 3.22). Deductions from such a compiled theory will give the same results as those from the former.

5.5 Limitations

In the last section, we have seen how the efficiency of query answering procedures for circumscription in ground theories can be improved. In this section, we return to the limitations of these procedures when they are applied to first-order theories.

The most important problem arises in the fact that both Przymusinski's [1989] and Ginsberg's [1989] algorithms are naive implementations of Theorem 5.2, which states the need for the existence of a certain conjunction G of clauses from $Newcarc(T, \neg F)$, ignoring that $Newcarc(T, \neg F)$ may contain many clauses. Therefore, they turn out to suffer from the following two problems in their computation:

1. Both "algorithms" do not work for first-order theories in general. There may be the case in which $Newcarc(T, \neg F)$ contains an *infinite number of* clauses. Even if the number of clauses in $Newcarc(T, \neg F)$ is finite, not all of them are the relevant parts needed to determine that F is a theorem of the circumscriptive theory. [6]

[6] While Przymusinski's prover does not compute all of $Newcarc(T, \neg F)$ but computes a precursor

2. Neither algorithm can handle the *answer extraction for open queries*. That is, when a query contains variables, the algorithms cannot return the substitution values of the variables for which the query holds. This is a much broader problem than "Yes/No" type questions.

A procedure attempting to solve the first problem is proposed by Poole [1989], which is a dialectical implementation of membership in all extensions. In [Helft *et al.*, 1989], a solution for the second problem was proposed, which finds a minimal, rather than maximal conjunction G of clauses from $Newcarc(T, \neg F)$. This solution can also solve the first problem, that is, it saves much of the unnecessary computation and provides answer extraction. To see this, the following two examples show that the idea of testing for the minimal disjunctions of explanations can save computation, as in the dialectical implementation of [Poole, 1989].

Example 5.10 [Helft *et al.*, 1989] Let T contain the following three clauses:

$$\forall x\,(\,N(x) \wedge \neg Ab(x) \supset G\,)\,,$$
$$\forall x\,(\,N(x) \supset N(s(x))\,)\,,$$
$$N(0)\,.$$

Consider whether the circumscription of Ab in T with variables N, G implies G or not. In this case, $Newcarc(T, \neg G)$ contains an infinite number of clauses:

$$Ab(0),\ \ Ab(s(0)),\ \ Ab(s(s(0))),\ \ \ldots$$

but anyone is enough to answer "Yes". Take, for example, the first one, $Ab(0)$: we can immediately see that the new characteristic clauses of $\neg Ab(0)$ are empty, that is, the negation of the explanation $\neg Ab(0)$ of G is unexplained, and the computation can stop. An attempt to compute all the new characteristic clauses of $\neg G$, or all explanations of G, would give no answer. Notice also that $\{Ab(0)\}$ is not a precursor of the others.

Example 5.11 [Helft *et al.*, 1989] Let T be the following:

$$Looks\_like\_Emu \wedge \neg Ab1 \supset Emu\,,$$
$$Looks\_like\_Ostrich \wedge \neg Ab2 \supset Ostrich\,,$$
$$Emu \supset Bird\,,$$
$$Ostrich \supset Bird\,,$$

of them, in some cases (like Example 5.10) the precursor may have potentially an infinite number of clauses for the first-order case.

$$\neg Emu \vee \neg Ostrich\,,$$
$$Looks\_like\_Emu\,,$$
$$Looks\_like\_Ostrich\,,$$
$$Looks\_like\_Bird \wedge \neg Ab3 \supset Bird\,,$$
$$Long\_Computation \supset Looks\_like\_Bird\,,$$
$$\cdots\ (lots\ of\ rules\ for\ Long\_Computation)\ \cdots$$

Here both $\neg Ab1$ and $\neg Ab2$ are explanations for $Bird$. Their negations are unexplained, and so $Bird$ follows from the circumscription. At this point, there is no need to look for other explanations for $Bird$; we can thus avoid the $Long\_Computation$ that would result from examining the remaining choice.

The solution to the second problem proposed by [Helft *et al.*, 1989] is further expanded in the subsequent sections, showing that queries can be answered by finding certain combinations of explanations. However, although we need not compute all of $Newcarc(T, \neg F)$ for computing an answer, a certain subset of them must be computed anyway. Therefore, the improvements proposed in the last section can still be applied to any proof procedure attempting to solve these problems.

5.6 Computing Answers for Circumscription

For the first-order predicate logic, techniques developed by Green [1969a; 1969b] are the basis for query-answering systems and are extensively used in deductive databases, logic programming and synthesis problems such as planning. These techniques rely on resolution-based theorem provers that attempt to obtain instances of the goal $G(\mathbf{x})$ by maintaining the information generated as a side effect during the proof of $\exists \mathbf{x}(G(\mathbf{x}))$. Theorem provers can decide whether a query follows from a given theory, and thus answer questions such as "Is there a coffee cup?"; the corresponding query-answering procedure computes the instance for which the query holds and can provide answers to questions such as "Where is the coffee cup?".

The latter half of this chapter addresses the query answering problem for logic databases augmented with a circumscription axiom [McCarthy, 1980; McCarthy, 1986] and related nonmonotonic formalisms. As in the first-order case, the answering procedure we present extracts information from a proof of the query, and are built on existing proof procedures for circumscription [Przymusinski, 1989; Ginsberg, 1989].

While existing theorem provers for circumscription can correctly answer whether or not a formula follows from a circumscription, Green's techniques, although necessary, are not sufficient to provide answer extraction. The reason is the following.

Circumscriptive theorem provers are based on finding explanations, or arguments, for the theorem to be proved, and showing that these explanations cannot be refuted. In general, many explanations compete in supporting the theorem, and a certain combination of these has to be found. We show that an informative answer corresponds to a particular combination of explanations, and present a procedure to find these combinations, together with some results that are useful to search this space and carry out significant pruning.

Although we focus on circumscription, the results we present obviously apply to its restrictions, such as logic databases using different types of closed-world assumptions, and similar default reasoning systems having explanation-based proof theories [Geffner, 1990; Poole, 1989].

5.6.1 Circumscription through Abduction

Here, we introduce a variation of the results in Section 5.2 based on an abductive framework, and provide the basic notion used in the rest of this chapter.

Definition 5.12 Let T be a theory, $CIRC(T;\mathbf{P};\mathbf{Z})$ its circumscription, F a formula.

1. $\mathbf{P}^- \cup \mathbf{Q}^{\pm}$ is called the *explanation vocabulary.*

2. A finite conjunction E of literals from the explanation vocabulary is an *elementary explanation for F relative to T* if

   (a) $T \cup \{E\} \models F$, and

   (b) $T \cup \{E\}$ is consistent.

   A disjunction of elementary explanations is called an *explanation.*

3. Let E be any explanation. An elementary explanation for $\neg E$ relative to T is called a *counter* to E.

4. If an explanation has no counters it is *valid.*

Note that here we extend the notion of *explanation* to more general one than an abductive framework given in Chapter 2. Each explanation defined in Definition 3.10 is called an *elementary explanation* in the following sections. The reason why we use a term *explanation* for a disjunction of (elementary) explanations is that if E_1 and E_2 are (elementary) explanations of F, then $E = E_1 \vee E_2$ satisfies the conditions: (a) $T \cup \{E\} \models F$, and (b) $T \cup \{E\}$ is consistent. Since such an E consists of the literals from the explanation vocabulary, we can consider it as a weaker explanation of F.

Theorem 5.13 (Variant of Theorem 5.2) $CIRC(T;\mathbf{P};\mathbf{Z}) \models F$ if and only if there exists a valid explanation for F relative to the theory T. □

Example 5.14 Consider the theory T consisting of the two formulas:

$$\forall x\,(Bird(x) \wedge \neg Ab(x) \supset Flies(x))\,,$$
$$Bird(Tweety)\,,$$

where $\mathbf{P} = \{Ab\}$, $\mathbf{Q} = \{Bird\}$ and $\mathbf{Z} = \{Flies\}$, so that the explanation vocabulary is $\{Ab\}^- \cup \{Bird\}^{\pm}$. Let us consider the query

$$F = Flies(Tweety).$$

Now, $\neg Ab(Tweety)$ is an explanation for F, as it implies F together with T and belongs to the explanation vocabulary. It has no counters, as no formula from the explanation vocabulary can be consistently added to T to deduce $Ab(Tweety)$. F is thus a theorem of $CIRC(T; Ab; Flies)$.

Next, let

$$T' = T \cup \{\, Ab(Tweety) \vee Ab(Sam)\,\}.$$

Then $\neg Ab(Sam)$ is a counter to $\neg Ab(Tweety)$ relative to T'. As no other valid explanation for F exists, F is not a theorem of the circumscription of Ab in T'.

5.6.2 Yes/No Answering Procedure

In line with the above results, the task of a query answering procedure for circumscription is to search for explanations of the query and test their validity.

To do so, the answering procedure can rely on an *explanation-finding algorithm*. Such an algorithm is provided with a set of clauses T, a clause F, and a vocabulary, and returns an elementary explanation E, that is, a conjunction of literals from the vocabulary, consistent with T, such that $T \cup \{E\} \models F$. Recall that elementary explanations can be obtained by computing the *new characteristic clauses*,

$$Newcarc(T, \neg F) = \mu\,[\,Th_{\mathcal{P}}(T \cup \{\neg F\}) - Th(T)\,]\,,$$

where the production field is set to $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^{\pm}\,\rangle$. Note that $Newcarc(T, \neg F)$ does not include those clauses implied by T alone, because their negations are inconsistent with T and they cannot be counters. The predicates of $\mathbf{P}$ have their sign changed because we look for the negation of E. The negation of each of such clauses (a conjunction of literals from the explanation vocabulary) is an elementary explanation (Proposition 3.11) and any disjunction of these is an explanation.

Testing the validity of an explanation represents the same computational problem: if an explanation E has no counter, then there is no new characteristic clause of E with respect to T and the production field $\langle \mathbf{P}^+ \cup \mathbf{Q}^\pm \rangle$ (Proposition 3.13). In symbols, given an algorithm to compute the set $Newcarc(T, F)$, we are interested in

$$Explanations(T, F) = \overline{Newcarc(T, \neg F)}$$

and

$$Valid(E, T) \Leftrightarrow Newcarc(T, E) = \emptyset.$$

In Section 5.4, we have already shown an explanation-finding algorithm, and it is not a concern from now on. The results we present concern how to *combine* explanations in order to extract answers from a proof. We thus can assume that such an algorithm exists and returns the correct explanations, and now concentrate on the query answering procedure. The following has been shown to correctly return "Yes/No" answers, and is used in [Ginsberg, 1989; Przymusinski, 1989].

Algorithm 5.15 (Yes/No Answering Procedure)

Step 1. (Generate Elementary Explanations)
Compute elementary explanations of F relative to T.

Step 2. (Combine Elementary Explanations)
Set the explanation E to the disjunction of all elementary explanations, and represent it in conjunctive normal form (i.e., as a set of clauses).

Step 3. (Test Validity)
Test if E has no counter, in which case answer "Yes"; otherwise answer "No".

This query answering procedure is not an exact implementation of Theorem 5.13 in one respect. The theorem stipulates the need for an arbitrary valid explanation, while the answering procedure, in Step 2, tests only one explanation for validity, namely the disjunction of all the elementary ones generated in Step 1. This is enough to return "Yes/No" answers, especially answers to ground queries. The reason is that if a certain disjunction of valid explanations exists, then the maximal disjunction is valid. This maximal disjunction is then tested for validity. The example we present next illustrates the inability of this procedure to provide answer extraction.

5.6.3 Example

I have to do some Prolog and Lisp programming this morning, and I need their manuals. Asking people around, I collect information about who has recently been using them. I know the office number of my colleagues, and I also know that normally people leave books in their offices. However, there are exceptions to this rule: for example, some of my colleagues work at home and don't bring back the books to the office. My knowledge can be expressed with the following theory T, where predicate symbols and constants have obvious intended interpretations:

$$\begin{gathered}
\forall x\, \forall y\, \forall z\ (\ \mathit{Had}(x, y) \wedge \mathit{Office}(x, z) \wedge \neg \mathit{Ab}(x) \supset \mathit{At}(y, z)\ ), \\
\mathit{Had}(\mathit{Fred}, \mathit{Prolog\text{-}manual}) \vee \mathit{Had}(\mathit{Mary}, \mathit{Prolog\text{-}manual}), \\
\mathit{Had}(\mathit{Harold}, \mathit{Prolog\text{-}manual}), \\
\mathit{Had}(\mathit{Kurt}, \mathit{Lisp\text{-}manual}), \\
\mathit{Office}(\mathit{Fred}, \mathit{EJ225}), \\
\mathit{Office}(\mathit{Mary}, \mathit{EJ230}), \\
\mathit{Office}(\mathit{Harold}, \mathit{EJ235}), \\
\mathit{Office}(\mathit{Kurt}, \mathit{EJ240}), \\
\forall x\, \forall y\, \forall z\ (\ \mathit{Different}(y, z) \supset \neg \mathit{At}(x, y) \vee \neg \mathit{At}(x, z)\ ), \\
\mathit{Different}(\mathit{EJ225}, \mathit{EJ230}) \wedge \cdots \\
\cdots \wedge \mathit{Different}(\mathit{EJ235}, \mathit{EJ240}).
\end{gathered}$$

Where should I look for the manuals? Suppose I submit to the theorem prover the query

$$F = \exists\, x\, y\ (\ \mathit{At}(x, y)\ )\,.$$

I am not really interested in knowing whether F is true or not. I would like to know how to get the manuals back, and do so without inspecting all the offices around.

If we set $\mathbf{P} = \{\mathit{Ab}\}$ and let the rest of the predicates vary, the minimal models of T can be divided in three groups, in each of which exactly one of $\neg \mathit{Ab}(\mathit{Harold})$, $\neg \mathit{Ab}(\mathit{Mary})$ or $\neg \mathit{Ab}(\mathit{Fred})$ is true. In the first group of these

$$\mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ235})$$

holds. In the second and the third groups,

$$\mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ225}) \vee \mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ230})$$

holds. Thus

$$\begin{gathered}
\mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ225}) \vee \mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ230}) \\
\vee\, \mathit{At}(\mathit{Prolog\text{-}manual}, \mathit{EJ235})
\end{gathered}$$

holds in all minimal models, and no subdisjunction does. Moreover, in all minimal models $\neg Ab(Kurt)$ is true, which means that

$$At(\textit{Lisp-manual}, EJ240)$$

is another theorem of the circumscription. These two smallest disjunction of answers provide me with information about where the manuals are.

However, Algorithm 5.15 produces the following.

Step 1. Three elementary explanations are computed:

$$\begin{aligned} E_1 &= \neg Ab(Fred) \wedge \neg Ab(Mary) \\ E_2 &= \neg Ab(Harold) \\ E_3 &= \neg Ab(Kurt) \end{aligned}$$

Step 2. The disjunction $E_1 \vee E_2 \vee E_3$ is transformed to conjunctive normal form. This is the conjunction of the following clauses:

$$\neg Ab(Fred) \vee \neg Ab(Harold) \vee \neg Ab(Kurt)$$
$$\neg Ab(Mary) \vee \neg Ab(Harold) \vee \neg Ab(Kurt)$$

Step 3. These clauses, when added to T, produce no new theorem in the vocabulary of positive Ab predicates, showing that the disjunction of explanations is valid, as it has no counters. The procedure correctly answers "Yes".

Instances of the query can never be returned with this procedure. The reason is that the actual substitution for the variable appearing in the query is lost in Step 2, when the explanation is converted from disjunctive to conjunctive normal form. The rest of this chapter describes a methodology and results on how to combine elementary explanations carefully in order to produce the informative answers.

5.7 Answer Extraction

We now consider the informative answers to a query F relative to a circumscription $CIRC(T; \mathbf{P}; \mathbf{Z})$ to be the most specific instances of F entailed by the circumscription; in the sequel we simply call them *answers* to the query.

Definition 5.16 Let $CIRC(T; \mathbf{P}; \mathbf{Z})$ be a circumscriptive theory, F an existentially quantified query. An *answer* to F is a formula A such that

1. $CIRC(T;\mathbf{P};\mathbf{Z}) \models A$,
2. A is a disjunction of instances of F, and
3. No A' not logically equivalent to A satisfies (1), (2) and $A' \models A$.

We now show how to produce such answers.

5.7.1 Obtaining Instances of the Query

As we noted in Section 5.6.2, we assume that an explanation-finding algorithm returns explanations for the query. The problem we address is that of finding answers, that is, the most specific disjunctions of instances of the query entailed by the circumscription. To compute such instances of the query, we use Green's techniques for first-order logic [Green, 1969a; Green, 1969b], which associate with F the clause

$$F' = \neg F \vee Ans(\mathbf{x})$$

where $\mathbf{x} = x_1, \ldots, x_n$ stands for the variables appearing in F. During the proof of F, Ans keeps track of the substitutions for which F holds at no extra cost. The explanation-finding algorithm is thus the same resolution-based algorithm used by existing theorem provers for circumscription [Ginsberg, 1989; Przymusinski, 1989], but instead of supplying the negation of the query F to compute $\overline{Newcarc(T, \neg F)}$, we supply F' and compute $Newcarc(T, F')$. Here, the positive occurrences of the answer predicate Ans are added to the production field $\langle\, \mathbf{P}^+ \cup \mathbf{Q}^{\pm} \,\rangle$. Accordingly, instead of obtaining (negations of) elementary explanations

$$E_1\,,\ \ldots\ ,\ E_n$$

for F, we obtain

$$E_1 \supset Ans_1\,,\ \ldots\ ,\ E_n \supset Ans_n\,,$$

where each Ans_i keeps the substitutions for a disjunction F_i of instances of F explained by E_i, that is,

$$T \cup \{E_i\} \models F_i \quad (i = 1, \ldots, n).$$

5.7.2 Computing Answers

Given a query, the explanation-finding algorithm can compute the candidate answers $F_1, F_2, \ldots$, the explanations for each of these $E_{11}, E_{12}, \ldots$, and their counters $C_{111}, C_{112}, \ldots$. An answer is a shortest disjunction of F_i that has a valid explanation, that is, for which at least one explanation exists with no counter.

Suppose that two instances F_1 and F_2 of the query have elementary explanations E_1 and E_2, neither of which is valid. E_1 and E_2 thus have counters C_1 and C_2. Now, $C = C_1 \wedge C_2$ is a candidate counter for $E = E_1 \vee E_2$ in the sense that

$$T \cup \{C_1 \wedge C_2\} \models \neg E$$

follows from $T \cup \{C_i\} \models \neg E_i$ for $i = 1, 2$. However, still C might not be a counter to E because, although each C_i is consistent with T, C might not be. If C is not consistent with T, and no other counter exists, E will be a valid explanation for $F_1 \vee F_2$.

Thus, while certain explanations may have counters, their disjunction might not, if the corresponding conjunction of potential counters is inconsistent. Therefore, compared with theorem provers that return "Yes/No" answers, *the only additional computation needed is the consistency check on combinations of counters.*

So the search problem consists of finding the disjunctions of instances of the query such that the potential counters of their explanations are inconsistent. A counter C is inconsistent with T if and only if $T \models \neg C$; and such counters belong to a particular vocabulary. It is thus rewarding to compute the following set of clauses.

Definition 5.17 Let $CIRC(T; \mathbf{P}; \mathbf{Z})$ be a circumscriptive theory. A *characteristic clause of* $CIRC(T; \mathbf{P}; \mathbf{Z})$ is a clause C_c that satisfies the following three:

1. Every literal of C_c belongs to $\mathbf{P}^+ \cup \mathbf{Q}^{\pm}$.
2. $T \models C_c$.
3. No clause C_c' not logically equivalent to C_c satisfies (1), (2) and $C_c' \models C_c$.

These are the restriction of the *prime implicates* [Reiter and de Kleer, 1987] of T to a particular vocabulary. They can be computed with the same linear resolution algorithm used by the explanation-finding procedure, as shown by [Inoue, 1991b].

We can take advantage of this set in at least the following two cases:

1. Let E be a minimal elementary explanation, and C a counter to E. Then

   $$T \cup \{C\} \models \neg E\,,$$

   and thus

   $$T \models \neg C \vee \neg E\,.$$

   Therefore, there is some $E' \models E$ such that $\neg C \vee \neg E'$ *is a characteristic clause of the circumscription.* In other words, to compute counters to an explanation, we consider the negation of its elementary components. The counters are the negation of the complement within a characteristic clause.

2. If $T \cup \{C\}$ is inconsistent, and thus $T \models \neg C$, then $\neg C$ *is implied by a characteristic clause of the circumscription.* This means that the consistency test on a combination of counters can be performed by entailment tests [7] on the characteristic clauses.

The above ideas can be implemented using different search strategies, which are not our direct concern. The following is a possible implementation of an algorithm [8] to return informative answers to a query in a circumscriptive theory.

Algorithm 5.18 (Query Answering Procedure)

Step 1. (Compilation)
Compute the characteristic clauses of the circumscription.

Step 2. (Generate Elementary Explanations)
Compute elementary explanations of F relative to T.

Step 3. (Compute Counters)
A counter to an explanation is the complement of its elementary components within the characteristic clauses. If an explanation has no counters, output the instance of the query explained by it.

Step 4. (Combinations of Counters)
Compute the conjunctions of counters whose negation is entailed by a characteristic clause. Such a conjunction of counters is inconsistent with T, and the corresponding disjunction of elementary explanations is valid. If such an explanation has no other counters, the corresponding disjunction of instances of the query is an answer. If the explanation has other counters, then they should be combined with other explanations to try to find an answer.

[7] We could define the characteristic cause of $CIRC(T; \mathbf{P}; \mathbf{Z})$ in Definition 5.17 by using the set closed under subsumption as in Definition 3.4 in Chapter 3 instead of using the entailment relation. Note that if C subsumes D then $C \models D$. Each "entailment test", then, could be replaced by a subsumption test.

[8] Strictly speaking, this is not an "algorithm". The reason is, in Step 1 or 2, it may produce an infinite number of characteristic clauses or elementary explanations. This procedure assumes that we have to compute elementary explanations using the "compiled" approach (of the terminology of [Reiter and de Kleer, 1987]). However, there is nothing to prevent us from using an "interpreted" approach; Steps 1 and 2 can be carried out in a demand driven manner for finding elementary explanations.

5.7.3 Example

We consider again the example of Section 5.6.3.

Step 1. Since Ab is the only minimized predicate, the only characteristic clause of the circumscription is:

$$Ab(Harold) \vee Ab(Fred) \vee Ab(Mary).$$

Step 2. We provide the explanation-finding algorithm with the clause

$$\neg At(x, y) \vee Ans(x, y),$$

and the answer predicate Ans.
Then we obtain the new clause

$$Ab(Fred) \vee Ab(Mary)$$
$$\vee Ans(Prolog\text{-}manual, EJ225) \vee Ans(Prolog\text{-}manual, EJ230),$$

indicating that

$$E_1 = \neg Ab(Fred) \wedge \neg Ab(Mary)$$

explains

$$A_1 = At(Prolog\text{-}manual, EJ225) \vee At(Prolog\text{-}manual, EJ230).$$

In a similar way,

$$E_2 = \neg Ab(Harold)$$

explains

$$A_2 = At(Prolog\text{-}manual, EJ235),$$

and

$$E_3 = \neg Ab(Kurt)$$

explains

$$A_3 = At(Lisp\text{-}manual, EJ240).$$

Step 3. $C_1 = \neg Ab(Harold)$, is a counter to E_1.
$C_2 = \neg Ab(Fred) \wedge \neg Ab(Mary)$, is a counter to E_2.
E_3 has no counters, thus A_3 is output.

Step 4. The only characteristic clause subsumes (in fact, is equivalent to) the disjunction of the negation of the two counters C_1 and C_2. This indicates that $T \cup \{C_1 \wedge C_2\}$ is inconsistent, and as $E_1 \vee E_2$ has no other counter, it is a valid explanation for $A_1 \vee A_2$.

5.8 Summary

We have compared two algorithms [Przymusinski, 1989; Ginsberg, 1989] to compute circumscription in ground theories relating them to abductive reasoning, and showing that they are based on the same theoretical results. We have also analyzed their computational properties, showing their different concerns: [Przymusinski, 1989] defines the set of formulas that needs to be computed and uses skip operations to compute them directly; [Ginsberg, 1989] concerns avoiding redundancy by recording the information during a deduction. The skip operation can be applied to other, more efficient versions of linear resolution, and further improvements on these methods can be incorporated into the procedure.

Nonmonotonic theorem provers often consist in a two-step classical deduction — making a default proof in which explanations are collected and checking validity of the explanations. We showed that the substitutions needed for query answering are lost in this process, and a combination of theses need to be found to produce the required answers. We presented a procedure for combining explanations in order to obtain informative answers, and results that enable an answering procedure to return the interesting answers with minimal search.

The importance of the results presented lies in their applicability to a wide class of systems that are either a restriction of circumscription, for example, databases using different types of closed-world assumptions (see [Przymusinski, 1989; Gelfond *et al.*, 1989]), or similar default reasoning systems having explanation-based proof theories [Poole, 1989; Geffner, 1990].

Chapter 6

Hypothetical Reasoning in Logic Programs

In order to express incomplete knowledge, extended logic programs have been proposed as logic programs with classical negation as well as negation as failure. This chapter discusses ways to deal with a broad class of commonsense knowledge by using extended logic programs. For this purpose, we present a uniform approach for dealing with both incomplete and contradictory programs, as a simple framework of hypothetical reasoning in which some rules are dealt with as candidate hypotheses that can be used to augment the background theory. This theory formation framework can be used for default reasoning, contradiction removals, the closed world assumption and abduction. We also show a translation of the theory formation framework to an extended logic program whose answer sets correspond to the consistent belief sets of augmented theories.

6.1 Introduction

Recent investigations in theories of logic programming have revealed a close relationship between the semantics of logic programs containing *negation as failure* and other theories of nonmonotonic reasoning developed in AI. Now, logic programming can be viewed as a simplified format of more general nonmonotonic logics such as *default logic* [Reiter, 1980] and *autoepistemic logic* [Moore, 1985]. While such a connection is useful for the better understanding of logic programs, it is not enough merely to define a logic program as a subset of a language capable of nonmonotonic reasoning: we would also like to know how such a language can be used for solving various problems and want to apply logic programming to *commonsense reasoning*. However, there have been few discussions as to how to represent knowledge of the sorts of problems

using logic programming (or other nonmonotonic logics). In this chapter, we will discuss a general framework based on logic programming for formalizing various forms of commonsense reasoning.

An important step for representation of commonsense knowledge is to allow *classical negation* ($\neg$) in logic programs (called *extended logic programs* [Gelfond and Lifschitz, 1990]) along with negation as failure (*not*). A semantics of an extended logic program can be given by its *answer sets*, which is a suitable extension of the *stable models* [Gelfond and Lifschitz, 1988] of a general logic program. As discussed by Gelfond and Lifschitz [1990], an important aspect of classical negation is that we can represent *incomplete information* by extended logic programs. This chapter considers further the usefulness of classical negation in logic programming, and present methods to deal with broader classes of commonsense knowledge.

The first point we should clarify is the role and effect of classical negation in various forms of reasoning. An extended logic program may be contradictory when both an atom A and its negation $\neg A$ are derivable, while a definite Horn program never causes a contradiction. That is, when we incorporate classical negation in programs, the notion of *consistency* becomes more important. We will summarize the notion, semantics and properties of classical negation in logic programming in Section 6.2.

Next, we will apply the notion of consistency to various kinds of reasoning in Section 6.3, focusing particularly on *default reasoning* (Section 6.3.1) and *abduction* (Section 6.3.4) as two basic forms of intelligent reasoning. We also consider *contradiction removals* (Section 6.3.2) and the *closed world assumption* (Section 6.3.3) as other important functions with which logic programming should cope. To deal with these applications, however, instead of showing individually how each style of reasoning can be expressed using extended logic programs, we shall propose a *uniform* framework for representing all of them. The basic idea behind this uniformity lies in two facts:

1. The notion of consistency mentioned above is also crucial for *theory formation*, that is, for selecting (or rejecting) appropriate (or inappropriate) theories that are formed from an incomplete description about the world by augmenting it with *hypotheses*.

2. As argued in studies of Theorist systems [Poole *et al.*, 1987; Goebel *et al.*, 1986; Poole, 1988] and others, the idea of theory formation is very useful for prototyping many AI reasoning systems by providing a simple "hypothesize-and-test" framework, that is, *hypothetical reasoning*. Then, hypothetical reasoning can be applied to both default and abductive reasoning.

Based on the idea of theory formation and hypothetical reasoning, we shall propose a very simple notion of *knowledge systems*. Formally, a knowledge system K is represented by a pair, (T, H), where

1. Each of T and H is an *extended logic program*,
2. T represents a set of *facts* [1] that are known to be true in the world, and
3. H represents a set of candidate *hypotheses* that may be expected to be true.

Then, the task of a knowledge system is *theory formation*, that is, to find a subset E of H such that $T \cup E$ is consistent. By using this mechanism, two types of reasoning mentioned above can be simply characterized as follows.

1. *Default reasoning.* Find a maximal subset E of H such that $T \cup E$ is consistent.
2. *Abduction.* Find an explanation $E\,(\subseteq H)$ of a formula O such that (i) $T \cup E$ is consistent and (ii) O is derived from $T \cup E$.

The above view of default reasoning as theory formation can be best seen in Poole's [1988] framework based on the Theorist system [Poole *et al.*, 1987]. The syntactical difference of our knowledge system from Poole's framework is that, while the latter uses first-order logic, ours uses extended logic programs as a "host" language. Then, for knowledge systems, the fact that a formula has an explanation does not imply that the formula holds in a maximally consistent augmentation (an example will be given in Example 6.27). In this sense, default reasoning is clearly distinguished from abduction.

We will also apply a knowledge system to the *closed world assumption*. As an example of extended logic programs, Gelfond and Lifschitz [1990] consider the inference of negation using the closed world assumption, which can be represented by augmenting the program with rules of the form

$$\neg A \leftarrow \mathit{not}\, A\,,$$

where A is any ground atom. However, they don't deal with them as hypotheses distinct from the program, but include them in the program. Suppose, for example, that the program consist of two rules:

$$Q \leftarrow \neg P(A), \neg P(B)\,,$$
$$\neg Q \leftarrow,$$

[1] In the logic programming community, "facts" traditionally mean atomic formulas asserted in a program (i.e., rules without bodies) and are distinguished from "rules" (with non-empty bodies). Here, however, any rule with or without body can be either a *fact* or a *hypothesis*. The distinction between facts and hypotheses is therefore determined not syntactically but by the user's intention.

and let us consider the closed world assumption for the predicate P:

$$\neg P(x) \leftarrow not\, P(x)\,.$$

If these rules are conjoined, no answer set is available. Instead, however, we would like to get two maximally consistent answer sets, $\{\neg P(A), \neg Q\}$ and $\{\neg P(B), \neg Q\}$, by dealing with these extra rules as distinct rules which can be invalidated or ignored when they cause inconsistencies. Furthermore, sometimes hypotheses may be added to make an incoherent program get consistent answer sets. Thus, the proposed framework can also be viewed as a system for *inconsistency resolution* or *contradiction removal*.

A naive computation to find a maximally consistent set of hypotheses would be carried out by searching through the power set of H, starting from the whole set H as the initial E and removing one rule from E at a time until we get consistent answer sets of $T \cup E$. In Section 6.4, we will show alternative methods for the computation by translating a knowledge system $K = (T, H)$ to an *extended logic program* K^* such that each answer set of K^* corresponds to an answer set of $T \cup E$ where E is a subset of H such that $T \cup E$ is consistent. Then, in Section 6.5, the transformed extended logic program K^* will be further transformed to a *general logic program*. Thus, through these transformations, any knowledge system can be characterized by an equivalent extended (or general) logic program.

As explained at the beginning of this section, this chapter is not intended to establish a further connection between general nonmonotonic reasoning formalisms and logic programming. Such a mathematical analysis is also an important issue, but it is not the subject of this chapter. Instead, we will show how our theory formation framework can explain uniformly various recent proposals of knowledge representation in logic programming. To verify this claim, the proposed framework will be compared to many other reasoning systems based on logic programming [Eshghi and Kowalski, 1989; Kakas and Mancarella, 1990; Gelfond, 1990; Kowalski and Sadri, 1990; Giordano and Martelli, 1990; Pereira *et al.*, 1991a; Goebel *et al.*, 1986; Baral *et al.*, 1991] in Section 6.6. In particular, the proposed method for inconsistency resolution is different from the TMS-style consistency maintenance. While the TMS adds a new rule to remove inconsistency, our proposal disregards a minimal set of hypotheses to remove incoherency. In this sense, the proposed framework can be considered as a generalization of nonmonotonic ATMSs [Dressler, 1989; Junker, 1989].

6.2 Background

This section presents basic properties of *extended logic programs* introduced by Gelfond and Lifschitz [1990], on which our theory formation framework is based.

6.2.1 Classical Negation

The most important innovation of extended logic programs is that *classical negation* is allowed to be used in programs. There are at least four reasons why classical negation is important for hypothetical reasoning.

1. The incorporation of classical negation into programs allows us to deal directly with incomplete information in representing knowledge as well as in answering queries. Then the *open world assumption* is employed for each incomplete definition of a predicate, but still we can freely specify the *closed world assumption* by using classical negation, as discussed by Gelfond and Lifschitz [1990] (see also Section 6.3.3).

2. The idea of allowing classical negation that appears in the heads of some rules is very useful to formalize *exceptions* to general rules. For instance, a study of representation of legislation by Kowalski [1989] suggests that it is natural to express legal rules using classical negation, and Kowalski and Sadri [1990] argue that a simple form of *default reasoning* can be dealt with very easily by using classical negation.

3. The introduction of classical negation provides us with the power to write contradictory rules. This is essential not only for *abduction* but for any kind of *theory formation* because, as argued by Goebel, Furukawa and Poole [1986], we are interested in systems that can deduce contradictions to *reject inconsistent theories*. This usage of classical negation is therefore similar to the notion of *integrity constraints*.

4. As we will see in Section 6.4, even in the notion of the simplest form of candidate hypotheses, that is, those in the form of ground atomic formulas, there appears to be the concept of classical negation. An atom A can be assumed to be true if it is consistent with a theory, that is, if $\neg A$ is not derived from a theory. This fact suggests that we may *transform each candidate hypothesis* into a rule using both classical negation and negation as failure.

6.2.2 Answer Sets

Now, we define the syntax and semantics of logic programs with classical negation in a way slightly extended from [Gelfond and Lifschitz, 1990].

Definition 6.1 An *extended logic program* is a set of *rules*, which are either of the form:

$$L \leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n\,, \tag{6.1}$$

or of the form:

$$\leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n\,, \tag{6.2}$$

where $n \geq m \geq 0$, and L and L_i's are *literals* each of which is either an atom A or its negation $\neg A$. For a rule of the form (6.1), L is called the *head* of the rule, and for a rule of the form either (6.1) or (6.2), $L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n$ is the *body* of the rule. Each rule of the form (6.2) is called an *integrity constraint*.

In the above definition, we allow *integrity constraints* as rules with empty heads, which are not explicitly defined in [Gelfond and Lifschitz, 1990], in a program. While there are several other logical specifications of integrity constraints (see [Reiter, 1990] for their differences), here we regard integrity constraints as *epistemic* or meta-level statements about "what a logic program should know" or "what should be true about a logic program" (see also Section 6.6.6). [2] Note that in the epistemic view, integrity constraints are usually defined as first-order formulas and are separated from other rules. However, every integrity constraint in the form of a first-order formula F can be first characterized as a rule without a head, $\leftarrow \mathit{not}\, F$, then translated into rules using the transformation of [Lloyd and Topor, 1984]. For instance, an integrity constraint $P \supset Q$ can be expressed by $\leftarrow P, \mathit{not}\, Q$. This translation is also employed by [Sadri and Kowalski, 1987]. In this way, we can include integrity constraints in a program so that we do not have to distinguish them from other rules.

The semantics of extended logic programs can be given in various ways: the *answer set semantics* [Gelfond and Lifschitz, 1990], the *extended stable semantics* [Przymusinski, 1990a], and others. In this chapter we choose the answer set semantics, although other semantics may be applicable to our framework of hypothetical reasoning. The choosing of the semantics will be reconsidered in Section 6.6.4. In the semantics of extended logic programs, a rule containing variables stands for the set of its ground instances. We denote by Lit the set of ground literals in the language. Then the semantics of an extended logic program is given by its *answer sets* in the following two steps.

Definition 6.2 Let Π be a set of ground rules not containing *not*. The *answer set*, $\alpha(\Pi)$, of Π is the smallest subset S of Lit satisfying the conditions:

1. For any rule $L \leftarrow L_1, \ldots, L_m$ in Π, if $L_1, \ldots, L_m \in S$, then $L \in S$;

[2] Note that Reiter considers a program as a set of first-order formulas and characterizes integrity constraints (and queries) as a set of epistemic formulas (called KFOPCE) [Reiter, 1990]. In our case, both a program and integrity constraints are expressed as extended logic programs. This is an extension of (monotonic) databases to (nonmonotonic) knowledge bases, whose semantics can be partially given by using Levesque's autoepistemic logic [Levesque, 1990]. The exact epistemic semantics of default logic and extended logic programs are considered in [Lifschitz, 1991].

2. For any integrity constraint $\leftarrow L_1, \ldots, L_m$ in Π, if $L_1, \ldots, L_m \in S$, then $S = Lit$;

3. If S contains a pair of complementary literals, then $S = Lit$.

Definition 6.3 Let Π be any extended logic program without variables. For any set $S \subseteq Lit$, let Π^S be the set of rules without *not* obtained from Π by deleting

1. every rule containing a formula $not\, L$ in its body with $L \in S$, and

2. every formula $not\, L$ in the bodies of the remaining rules.

Then, S is an *answer set* of Π if S is the answer set of Π^S, that is, $S = \alpha(\Pi^S)$.

The effects of introducing integrity constraints and classical negation into a program appear in the second and third conditions of Definition 6.2. Intuitively speaking, each answer set is a possible set of beliefs: each literal in an answer set can be considered to be true in the belief set. Since positive and negative literals have the same status, the result of negation by failure to prove an atom A does not mean that its negation $\neg A$ is true. If an answer set contains neither A nor $\neg A$,[3] the truth value of A is *unknown* in the belief set. Thus, the answer set semantics can provide for indefinite answers in answering queries, and such unknown information can be referred to in an extended disjunctive database.

If Π is a *general* logic program, i.e., a set of rules without classical negation, then the answer sets of Π are identical to the *stable models* of Π given by Gelfond and Lifschitz [1988], except that Π may have the answer set Lit if no other answer set satisfies the integrity constraints in Π.

6.2.3 Consistency

As explained in Section 6.1, the notion of *consistency* is important for extended logic programs and for theory formation. Here, we classify extended logic programs as follows.

Definition 6.4 Let Π be an extended logic program.
(1) Π is *consistent* if it has a consistent answer set.
(2) Π is *contradictory* if it has an inconsistent answer set.
(3) Π is *incoherent* if it has no answer set.

[3] This is a big difference from well-founded semantics [van Gelder *et al.*, 1991] or stationary semantics [Przymusinski, 1990b]: we do not allow the inference that if A does not match the head of any rule of Π in accordance with the default reasoning behind negation as failure, then put A into the false part.

We see that there are two kinds of *inconsistency*: *contradiction* (2) and *incoherency* (3). The above definition is exclusive and complete: every program is either consistent, contradictory, or incoherent. This is verified by the following two observations.

Proposition 6.5 (*Minimality of answer sets* [Gelfond and Lifschitz, 1990]) Let Π be an extended logic program. For any two answer sets S and S' of Π, if $S \subseteq S'$ then $S = S'$.

Corollary 6.6 No extended logic program is both consistent and contradictory, and a contradictory program has the unique answer set Lit.

Gelfond and Lifschitz [1990] show the relation between the answer sets of an extended logic program and the *extensions* of the corresponding Reiter's default theory [1980]. Every rule in an extended logic program Π of the form

$$L \leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n$$

can be identified with the *default rule*

$$\frac{L_1 \wedge \ldots \wedge L_m \ : \ \mathsf{M}\,\overline{L_{m+1}}, \ldots, \mathsf{M}\,\overline{L_n}}{L},$$

where $\overline{L}$ is the literal complementary to L: when A is an atom, $\overline{A} = \neg A$ and $\overline{\neg A} = A$. According to [Gelfond and Lifschitz, 1990], there is a 1-1 correspondence between the answer sets of Π and the extensions of the default theory Π. Thus, we can discuss the *consistency* of a program by means of its answer sets; a program has a consistent answer set if and only if the corresponding theory has a consistent extension. Note that a rule not containing *not*

$$L \leftarrow L_1, \ldots, L_m \quad \text{or} \quad \leftarrow L_1, \ldots, L_m \tag{6.3}$$

can be identified with the default rule

$$\frac{L_1 \wedge \ldots \wedge L_m \ :}{L} \quad \text{or} \quad \frac{L_1 \wedge \ldots \wedge L_m \ :}{\mathit{false}} \tag{6.4}$$

respectively, and that the rules without bodies in Π can be represented by

$$W_\Pi = \left\{ \frac{:}{L} \;\middle|\; (L \leftarrow) \in \Pi \right\}.$$

While the default rules of the form (6.4) are not excluded by Reiter's definition, the existence of at least one justification for each default rule is presupposed in [Reiter,

1980, Corollary 2.2], which implies that a closed default theory Π has an inconsistent extension if and only if W_Π has an inconsistent extension. [4] In this case, however, a default theory Π may have an inconsistent extension even though the set of literals appearing in W_Π is consistent. The precise characterization of contradictory programs can be given simply as follows.

Proposition 6.7 An extended logic program Π is contradictory if and only if the set of rules of the form (6.3) (i.e., the rules without *not*) in Π is contradictory.

> **Proof:** *Lit* is an answer set of Π if and only if *Lit* is the answer set of Π^{Lit} that is the set of rules obtained from Π by deleting every rule containing a formula $not\, L$ in its body (by Definition 6.3) if and only if Π^{Lit} has an inconsistent answer set (by Definition 6.2). □

The above proposition tells us that for a contradictory program, the inconsistent augmentation occurs only when the rules without negation as failure are inconsistent. On the other hand, we can see that the source of an incoherency always lies in a set of rules containing *not* in a program. These properties of inconsistencies will be taken into account when we characterize the consistent subsets of an inconsistent program for the transformation of our theory formation framework into a single extended logic program in Section 6.4.

6.3 Theory Formation

In this section, we give a simple framework for hypothetical reasoning based on extended logic programs and its application to various forms of reasoning. The uniform framework we consider is based on a meta-theoretical partition of knowledge. That is, when our whole knowledge is given as a set Π of rules in an extended logic program, Π is split into two parts (T, H) such that $T \cup H = \Pi$ and $T \cap H = \emptyset$, where T stands for a set of facts and H for a set of candidate hypotheses that may be expected to be true. The resulting system is called a *knowledge system*.

Definition 6.8 A *knowledge system* is a pair (T, H), where T and H are extended logic programs. For a knowledge system $K = (T, H)$, the set T is called the *facts* of K, and the set H is called the (*candidate*) *hypotheses* of K.

[4] As far as the author knows, this observation for *justification-free default rules* was first discussed by Brewka [1989a]. A default rule of the form (6.4) cannot be replaced with

$$\frac{L_1 \wedge \ldots \wedge L_m \ :\ \mathsf{M}\, true}{L},$$

because such a transformation would never produce inconsistent extensions.

As explained in Section 6.1, the main task of a knowledge system is *theory formation*, that is, to find a subset E of H such that $T \cup E$ is consistent. Here, we first characterize a condition under which a knowledge system can perform theory formation.

Definition 6.9 Let $K = (T, H)$ be a knowledge system. K is *consistent* if there is a set $E \subseteq H$ such that $T \cup E$ is consistent. K is *contradictory* if for any set $E \subseteq H$, $T \cup E$ is contradictory. K is *incoherent* if K is neither consistent nor contradictory.

The above definition says that theory formation can be achieved only by a consistent knowledge system.

Proposition 6.10 Let $K = (T, H)$ be a knowledge system.
(1) K is contradictory if and only if T is contradictory.
(2) If T is consistent then K is consistent.
(3) If K is incoherent then T is incoherent.

> **Proof:** (1) The only-if-part is obvious from Definition 6.9. The if-part is a direct consequence of Proposition 6.7.
> (2) Obvious from Definition 6.9 since $E = \emptyset$ satisfies the condition.
> (3) We prove the contrapositive of the claim. Suppose that T is not incoherent, that is, T is either consistent or contradictory. If T is consistent, then K is consistent by (2). If T is contradictory, then K is contradictory by (1). In both cases, K is not incoherent. □

The above proposition tells us that a contradiction may not be removed by an addition of rules to the program. The converse directions of Proposition 6.10 (2) and (3) do not hold. Adding hypotheses to an incoherent program may make the knowledge system consistent.

Example 6.11 Let us consider the knowledge system (T, H) where $T = \{ P \leftarrow not\, P \}$ and $H = \{ P \leftarrow \}$. While T is incoherent, $T \cup H$ has a consistent answer set $\{P\}$.

In the following subsections, we will consider formalizations for several kinds of commonsense reasoning by using theory formation.

6.3.1 Default Reasoning

To augment an incomplete description of knowledge, one wants to represent and use "default" and "prototypic" knowledge. *Default reasoning* is thus one of the most obvious and important applications of theory formation, where default knowledge is taken into account unless there is evidence to the contrary. However, since default knowledge is usually inconsistent as a whole, simply adding all default knowledge to the background program would often result in no consistent answer set. To overcome this difficulty, we can deal with default knowledge as candidate hypotheses, and use them to augment the theory and to predict what we expect to be true. Then, as many hypotheses as possible are taken into account in an augmented set of beliefs. The notion of such an answer set of an augmented theory by a maximally consistent set of hypotheses roughly corresponds to a *Theorist extension* [Poole, 1988].

Definition 6.12 Let $K = (T, H)$ be a knowledge system. An *extension base* of K is an answer set of $T \cup E$ where E is a maximal (with respect to set inclusion) subset of H such that $T \cup E$ is consistent.

For default reasoning, the task of a knowledge system is to get its extension bases. This framework can make a contradictory or incoherent program Π become a consistent knowledge system (T, H) such that $\Pi = T \cup H$, provided that *prototypic* or *typical* knowledge is appropriately put into a set H of default hypotheses that is distinct from a set T of rules representing *factual* or *exceptional* knowledge. To obtain extension bases, some default hypotheses are allowed to be ignored, but no hypothesis can be dispensed with unless it is necessary to do so.

Example 6.13 Suppose we have the knowledge system $K = (T, H)$, where

$$\begin{array}{rll} T = \{ & \neg Flies(x) \leftarrow Penguin(x)\,, & \\ & Bird(x) \leftarrow Penguin(x)\,, & \\ & Bird(Polly) \leftarrow\,, & \\ & Penguin(Tweety) \leftarrow & \} \,, \\ H = \{ & Flies(x) \leftarrow Bird(x) & \} \,. \end{array}$$

Here it is easy to see that $T \cup H$ is contradictory. The unique extension base of K is

$$\begin{array}{l} S = \{\, Bird(Polly),\, Penguin(Tweety), \\ \quad Bird(Tweety),\, Flies(Polly),\, \neg Flies(Tweety) \,\} \,. \end{array}$$

Notice that the hypothesis is used for $x = Polly$ but is ignored for $x = Tweety$.

There are many issues addressed in default reasoning, such as multiple extension problems and priorities of default knowledge. Thanks to the two connectives in logic programming, the nonmonotonic operator *not* and the constructive implication $\leftarrow$, some of these topics are dealt with more naturally by our framework than by systems using first-order logic as a knowledge representation language. [5] We will not pursue these topics further because they are beyond the scope of this chapter. Instead, we will show in the next two subsections that maximally consistent theories of knowledge systems can be used for providing the meanings of extended logic programs representing tasks other than default reasoning.

6.3.2 Inconsistency Resolution

Suppose that an extended logic program Π is inconsistent: either incoherent or contradictory. In this case, we cannot get any information on beliefs because either Π has no answer set or the answer set contains every literal. Instead of simply noticing that the program has bugs, we would like to have some conclusions about objects that are irrelevant to the inconsistency. Reasoning in the presence of inconsistency is thus often necessary in database systems or expert systems, and has been studied in the field of non-standard logics in an attempt to restrict inferences from contradictions. The main difficulty here is that an incoherent program does not infer anything, so such a restriction does not work. That is why we need another approach to this problem.

If Π represents a finite set of rules, we may characterize maximally consistent subsets of Π. If a belief is contained in an answer set of such a maximal subset, we conjecture that the belief may be true in a possible belief set of Π if inconsistencies are removed. The idea of isolating inconsistencies is not a new one; for example, it can be seen in Rescher [1964] for propositional logic. When there are multiple maximally consistent subsets of Π, the user may be responsible for choosing one of them. More formally, for Π, we shall consider the knowledge system

$$K = (\emptyset, \Pi)\,.$$

Each extension base of K can give the meaning of a result of reasoning (with inconsistency) from Π. If Π is an incoherent program, then by considering the corresponding knowledge system K, for each source of incoherencies called "odd-loops", a rule in the loop will be removed to obtain consistency.

[5] Another feature of using $\leftarrow$ is that, while Poole's [1988] system needs constraints to prevent the use of contrapositives of clauses, they are not necessary for extended logic programs (see [Kowalski and Sadri, 1990]). See also the difference between the two approaches with respect to naming hypotheses in Section 6.4.2.

Example 6.14 Consider the knowledge system $K = (\emptyset, \Pi)$ where Π is given as:

$$P \leftarrow not\, P\,,$$
$$Q \leftarrow not\, R\,,$$
$$R \leftarrow not\, S\,.$$

The unique extension base of K is $\{ R \}$. No rules other than the first can be disregarded; any set of rules without either the second or third rule would remain incoherent unless the first rule was removed.

If some of the rules of Π are reliable and some are suspect, then of course we may have a knowledge system like $K = (\Pi_1, \Pi_2)$, where Π_1 is knowledge that we are completely confident about, and Π_2 is knowledge that is less certain. We will discuss this issue later more generally in Section 6.6.6.

Example 6.15 (Barber's Paradox) Suppose that Π is the extended logic program consisting of two rules:

$$Shaves(Noel, x) \leftarrow not\, Shaves(x, x)\,,$$
$$\neg Shaves(Casanova, Casanova) \leftarrow\ .$$

This Π is incoherent because the rule

$$Shaves(Noel, Noel) \leftarrow not\, Shaves(Noel, Noel) \tag{6.5}$$

is present in the program. Now, let $K = (T, H)$ be a knowledge system where

$$\begin{aligned} T &= \{\ \neg Shaves(Casanova, Casanova) \leftarrow \}\,, \\ H &= \{\ Shaves(Noel, x) \leftarrow not\, Shaves(x, x)\ \}\,. \end{aligned}$$

The hypothesis (6.5) is ignored in the unique extension base of K,

$$\{\ \neg Shaves(Casanova, Casanova),\ Shaves(Noel, Casanova)\ \}.$$

6.3.3 Closed World Assumption

Another interesting application along the line of maximally consistent theory formation is the *closed world assumption* (CWA) [Reiter, 1978]. For a ground atom A, it is very useful to define a *CWA rule for* A as a rule with the head $\neg A$ in an extended logic program. For instance, Gelfond and Lifschitz [1990] define a *(Reiter's) CWA rule* for each predicate P with n distinct variables $\mathbf{x} = x_1, \ldots, x_n$ in the language as

$$\neg P(\mathbf{x}) \leftarrow not\, P(\mathbf{x})\,. \tag{6.6}$$

Then, given a *general logic program* Π, an extended logic program $\Pi \cup CW$ where CW is the set of CWA rules (6.6) for all predicates precisely characterizes the meaning of Π within the stable model semantics [Gelfond and Lifschitz, 1990, Proposition 4]. The CWA rules for all (or some) predicates in this case are consistent with any coherent general logic program.

For extended logic programs, while an incomplete specification of knowledge often involves the *open world assumption* (OWA) for some predicates, we would like to consider the CWA for other predicates, as argued by Gelfond and Lifschitz. However, as shown in Section 6.1, if Reiter's CWA rules (6.6) are used together with the OWA for an *extended logic program*, then the augmented program is not consistent in general. Thus, we should not assume all negative ground literals even if each of them can be consistently assumed. This problem is analogous to the application of the CWA for non-Horn clauses in a database, which may produce an inconsistent augmentation [Reiter, 1978]. For a disjunctive database, Minker [1982] proposes the *generalized closed world assumption* (GCWA) which derives $\neg A$ for a ground atom A if A is true in no minimal model of the clauses. For an extended logic program, we should generalize the GCWA since the minimal models are not sufficient for characterizing the program. [6] In this case, however, we cannot simply derive $\neg A$ even if no answer set of the program contains A. For example, while a program $\{ Q \leftarrow \neg P \}$ has the unique answer set $\emptyset$ in which neither P nor Q is contained, assuming both $\neg P$ and $\neg Q$ causes a contradiction. Therefore, we propose to test each ground atom A for the membership in the extension bases of a knowledge system as follows.

Definition 6.16 Let Π be an extended logic program, and $P^{\neg}$ a set of some CWA rules. We denote by $CWA(\Pi, P^{\neg})$ the set of literals contained in every extension base of the knowledge system $(\Pi, P^{\neg})$. We say a literal L is *derived from* Π *under the CWA rules* $P^{\neg}$ (denoted as $CWA(\Pi, P^{\neg}) \models L$) if $L \in CWA(\Pi, P^{\neg})$.

This definition preserves the consistency of an extended logic program when it is augmented with CWA rules.

Proposition 6.17 Let Π be a consistent extended logic program and $P^{\neg}$ a set of some CWA rules. Then, $CWA(\Pi, P^{\neg})$ is consistent.

> **Proof:** Since Π is consistent, each extension base S of $(\Pi, P^{\neg})$ is obviously consistent with Π. Then, for any ground atom A, S does not contain both A and $\neg A$. Since $CWA(\Pi, P^{\neg})$ is the intersection of all extension bases, the result follows. □

[6] For a different purpose, Gelfond [1991] also generalizes the GCWA in the context of extended disjunctive databases [Gelfond and Lifschitz, 1991]. See Section 6.6.5.

We shall consider two kinds of specification of CWA rules $P^{\neg}$ for an extended logic program Π. The first one is to use a knowledge system

$$K_1 = (\Pi, CW)\,,$$

where CW is CWA rules (6.6) for some predicates.

Example 6.18 Let the extended logic program Π consist of the following three rules:

$$\begin{aligned} Q &\leftarrow \neg P(A)\,,\\ \neg Q &\leftarrow \neg P(B)\,,\\ P(C) &\leftarrow P(A),\, P(B)\,. \end{aligned}$$

And suppose that CW consists of the CWA rule for the predicate P:

$$\neg P(x) \leftarrow not\, P(x)\,.$$

It is easy to see that $\Pi \cup CW$ is incoherent. Now consider the knowledge system $K_1 = (\Pi, CW)$. There are two extension bases of K_1:

$$\{\, \neg P(A),\, \neg P(C),\, Q \,\} \text{ and } \{\, \neg P(B),\, \neg P(C),\, \neg Q \,\}\,.$$

Since $\neg P(C)$ is contained in both extension bases, it holds that $CWA(\Pi, CW) \models \neg P(C)$.

The second method is simpler than the first and uses a knowledge system

$$K_0 = (\Pi, NL)\,,$$

where NL is CWA rules of the form

$$\neg P(\mathbf{x}) \leftarrow \tag{6.7}$$

for some predicates with distinct variables $\mathbf{x}$.

Example 6.19 Let the extended logic program Π consist of the following rules:

$$\begin{aligned} P &\leftarrow not\, Q\,,\\ Q &\leftarrow not\, R\,. \end{aligned}$$

Π has the unique answer set $S = \{Q\}$. Now, suppose that NL contains three assertions,

$$\begin{aligned} \neg P &\leftarrow\,,\\ \neg Q &\leftarrow\,,\\ \neg R &\leftarrow\ . \end{aligned}$$

The unique extension base of the knowledge system $K_0 = (\Pi, NL)$ is

$$S' = S \cup \{ \neg P, \neg R \} = \{ \neg P, Q, \neg R \}.$$

In this case, Π is a general logic program, and the extension base S' is the same as the answer set of the program $\Pi \cup CW$, where CW is the Reiter's CWA rules of the form (6.6).

Proposition 6.20 Let Π be a general logic program. Suppose that CW^* is the CWA rules (6.6) for all predicates, and that NL^* is the rules (6.7) for all predicates. Then, the following three are equivalent.
(1) S is a consistent answer set of $\Pi \cup CW^*$.
(2) S is an extension base of the knowledge system (Π, CW^*).
(3) S is an extension base of the knowledge system (Π, NL^*).

Proof: (1) $\Rightarrow$ (2). Since $\Pi \cup CW^*$ is consistent by the supposition, obviously its answer set S is an extension base of (Π, CW^*).
(2) $\Rightarrow$ (3). This is a special case of Proposition 6.31, which will be described in Section 6.4.1, where H_0 and H_1 in Proposition 6.31 correspond to NL^* and CW^*, respectively.
(3) $\Rightarrow$ (1). Since Π does not contain classical negation, it holds that Π is consistent if (Π, NL^*) is consistent. Now, an extension base S of (Π, NL^*) can be represented by the set of literals consisting of the literals of a stable model S_0 of Π together with the negative literals each of whose atom is not contained in S_0. By [Gelfond and Lifschitz, 1990, Proposition 4], this S is an answer set of $\Pi \cup CW^*$. □

Note again that for an extended logic program Π, Proposition 6.20 does not hold. The differences between the two knowledge systems, (Π, CW) and (Π, NL), appear in the next two examples, which show that (Π, CW) *is more appropriate* for specifying the closed world assumption for extended logic programs.

Example 6.21 Let us consider the following extended logic program Π and two sets of hypotheses, CW and NL:

$$\Pi = \{ \quad P \leftarrow not\, \neg P \quad \},$$
$$CW = \{ \quad \neg P \leftarrow not\, P \quad \}, \qquad NL = \{ \quad \neg P \leftarrow \quad \}.$$

There are two extension bases of $K_1 = (\Pi, CW)$: $S_1 = \{P\}$ and $S_2 = \{\neg P\}$. Hence, $CWA(\Pi, CW) \not\models \neg P$. But only S_2 is the unique extension base of $K_0 = (\Pi, NL)$, thus $CWA(\Pi, NL) \models \neg P$. Thus, (Π, NL) makes as many negative literals hold as possible, while (Π, CW) preserves the semantics of Π in a possible belief set. Obviously, $\neg P$ should not be derived under the CWA since the unique answer set of Π contains P.

Example 6.22 Suppose that Π, CW and NL are given as follows.

$$\begin{array}{rl} \Pi = \{ & P \leftarrow not\, \neg P\,, \\ & \leftarrow \neg P, \neg Q \quad \}\,, \\ CW = \{ & \neg P \leftarrow not\, P\,, \\ & \neg Q \leftarrow not\, Q \quad \}\,, \end{array} \qquad \begin{array}{rl} NL = \{ & \neg P \leftarrow\,, \\ & \neg Q \leftarrow \quad \}\,. \end{array}$$

There is only one extension base of $K_1 = (\Pi, CW)$: $S_1 = \{ P, \neg Q \}$, which is the unique answer set of $\Pi \cup CW$. Hence, $CWA(\Pi, CW) \models \neg Q$. However, unlike the previous example, $K_0 = (\Pi, NL)$ obtains an extra extension base in this case: $S_2 = \{ \neg P \}$. Therefore, $CWA(\Pi, NL) \not\models \neg Q$. Since the unique answer set of Π is $\{ P \}$ and thus $\neg P$ should not derived under the CWA, we would not have any reason to reject the assumption about $\neg Q$.

Proposition 6.23 Let Π be a consistent extended logic program, CW a set of CWA rules (6.6), and A any ground atom such that the rule $\neg A \leftarrow not\, A$ is in CW. If A is in some answer set of Π, then $CWA(\Pi, CW) \not\models \neg A$.

> **Proof:** Suppose that $CWA(\Pi, CW) \models \neg A$. Let S be any extension base of (Π, CW) and E a maximal subset of CW such that S is an answer set of $\Pi \cup E$. By the definition of $CWA(\Pi, CW)$, S contains $\neg A$ and thus does not contain A. Then, the CWA rule $(*)$ is in E. Now, assume that there exists an answer set S' of Π such that S' contains A. Then, adding the CWA rule $(*)$ to Π does not prevent A from being in an answer set of the augmented program. Therefore, the CWA rule $(*)$ cannot be in E, contradiction. Hence, no answer set of Π contains A, and the result follows. $\Box$

The above proposition shows that the knowledge system (Π, CW) agrees with our intuition of the closed world assumption for extended logic programs.

6.3.4 Abduction

Theory formation was originally motivated by the goal of providing a formal account of inference to the best explanation of observations. This inference has been known as *abduction*, which plays a fundamental role in commonsense reasoning such as diagnosis, synthesis, intelligent databases, sophisticated user interfaces and communication among intelligent agents. Also, as argued by Kowalski [1990], abduction will be one of the major promising extensions of logic programming. For abduction, a knowledge system can be used to find an *explanation* E of a given formula O as follows.

Definition 6.24 Let $K = (T, H)$ be a knowledge system, and O be a formula. A set $E \subseteq H$ is an *explanation of* O *(with respect to* K*)* if:

1. $T \cup E$ is consistent, and
2. O is *derived from* $T \cup E$.

While the first condition is clear, the meaning of the second condition is somewhat controversial. It may be expressed in either of the following:

(a) There is an answer set of $T \cup E$ which satisfies O.

(b) O is satisfied by every answer set of $T \cup E$.

Here, we assume that O is simply a conjunction of literals, and we say that O is *satisfied* by a set $S \subseteq Lit$ if every literal in O is contained in S. [7] We write E $explains_1$ O if E is an explanation of O in the sense of the first definition of derivability (a), and write E $explains_2$ O if E is an explanation of O in the sense of the second definition of derivability (b).

Unlike Poole's [1988] system, *semimonotonicity* [Reiter, 1980] does not hold even if either all candidate hypotheses in a knowledge system are rules without bodies or they can be identified with Reiter's *normal default rule* $L \leftarrow L_1, \ldots, L_m, not\, \neg L$. In other words, for two knowledge systems $K = (T, H)$ and $K' = (T, H')$ such that $H' \subseteq H$, the fact that S' is an extension base of K' does not imply that there is an extension base S of K such that $S' \subseteq S$. This is because the rules T can be identified with Reiter's *nonnormal* default rules.

Example 6.25 Let $K = (T, H)$ and $K' = (T, H')$ be two knowledge systems where

$$\begin{array}{rll} T = \{ & P \leftarrow B\,, & \\ & Q \leftarrow A,\, not\, P\,, & \\ & P \leftarrow A,\, not\, Q & \}\,, \\ H = \{ & A \leftarrow, & \\ & B \leftarrow & \}\,, \\ H' = \{ & A \leftarrow & \}\,. \end{array}$$

K' has two extension bases: $S_1 = \{ A, Q \}$ and $S_2 = \{ A, P \}$. Clearly $H' \subseteq H$. However, S_1 is not a subset of the unique extension base of K: $S = \{ A, B, P \}$.

[7] We can reduce the restriction that an explained formula O is a conjunction of literals. If O is any formula, then we can regard it as an integrity constraint (like [Reiter, 1990]) so that it should be satisfied by an answer set. Since this integrity constraint can be transformed into a rule $\leftarrow not\, O$ (see Section 6.2.2), its satisfaction can be checked by finding an answer set of the extended logic program with such rules.

From the above discussion, the fact that a formula has an explanation does not imply that the formula is satisfied by an extension base. That is, explicability and membership in an extension differ for knowledge systems. In this sense, default reasoning is clearly distinguished from abduction. In default reasoning, a set H of hypotheses is used as *defaults*, whereas in abduction it is used as *premises*. If H represents a set of defaults, then an explanation is acceptable only when it is included in a maximal subset E of H such that $T \cup E$ is consistent. In other words,

Definition 6.26 Let $K = (T, H)$ be a knowledge system and O a formula. Assume that H represents a set of *defaults*. A set $E \subseteq H$ *predicts* O *(with respect to* K*)* if:

1. E *explains*$_\mathbf{1}$ O, and
2. there is a set $E' \supseteq E$ satisfying the conditions:
   (a) E' is a maximal subset of H such that $T \cup E'$ is consistent, and
   (b) O is derived from $T \cup E$.

For the condition (2), if E' *explains*$_\mathbf{1}$ O, then we write E *predicts*$_\mathbf{1}$ O; else (that is, if E' *explains*$_\mathbf{2}$ O) we write E *predicts*$_\mathbf{2}$ O.

Example 6.27 Let $K = (T, H)$ be the same knowledge system as Example 6.25, that is,

$$\begin{array}{rll} T = \{ & P \leftarrow B\,, & \\ & Q \leftarrow A,\, not\, P\,, & \\ & P \leftarrow A,\, not\, Q & \}\,, \\ H = \{ & A \leftarrow, & \\ & B \leftarrow & \}\,. \end{array}$$

1. $E_1 = \{ A \leftarrow \}$. $T \cup E_1$ has two answer sets: $S_1 = \{ A, Q \}$ and $S_2 = \{ A, P \}$.
   E_1 *explains*$_\mathbf{1}$ both Q and P, but cannot *explain*$_\mathbf{1}$ $P \wedge Q$.
   E_1 *explains*$_\mathbf{2}$ neither Q nor P.
2. $E_2 = \{ B \leftarrow \}$. $T \cup E_2$ has the unique answer set: $S_3 = \{ B, P \}$.
   E_2 *explains*$_\mathbf{1}$ and *explains*$_\mathbf{2}$ P.
3. $H = E_1 \cup E_2$. K has the unique extension base: $S = \{ A, B, P \}$.
   H *explains*$_\mathbf{1}$ and *explains*$_\mathbf{2}$ P.
4. Each of E_1, E_2, and H *predicts*$_\mathbf{1}$ and *predicts*$_\mathbf{2}$ P.
   Q can be neither *predicted*$_\mathbf{1}$ nor *predicted*$_\mathbf{2}$ by any set of hypotheses.

If we follow the first definition of derivability, Q has an explanation E_1 because S_1 contains Q. However, since S_1 is not a subset of the unique extension base S of K, Q does not hold in an extension. Notice that in this case E_1 can also explain P because S_2 contains P. It is curious that $P \wedge Q$ cannot be explained by E_1 while E_1 can explain both P and Q.

If we use the second definition of derivability, Q can never be explained with respect to K because S_2 does not satisfy Q. In this case, P cannot be explained by E_1 either, but P can be explained by either E_2 or H.

Since $T \cup H$ is consistent, if H represents default knowledge, then P can be predicted, but Q cannot be predicted, whichever definition of derivability we choose.

6.4 Reduction to Extended Logic Programs

In this section, we will show a method of the transformation from any knowledge system $K = (T, H)$ to a *single* extended logic program K^* such that every consistent answer set of $T \cup E$ for any $E \subseteq H$ can be characterized by an answer set of K^*, and vice versa. Also, the extension bases of K will be shown to correspond to the set of answer sets of K^* with some property. Moreover, we will show in Section 6.5 that this program K^* can be further reduced to a general logic program with integrity constraints. In other words, we will show that there is a 1-1 correspondence between the semantics of knowledge systems and the stable model semantics of their transformed general logic programs. Thus, the transformation method we present has the advantage that it gives an easy way to understand the semantics of knowledge systems: since the disjoint set of candidate hypotheses (and classical negation) are eliminated in the process of transformation, we can apply the answer set (or stable model) semantics to the resulting extended (or general) logic program.

Recall that even if a program Π is incoherent, an augmented program $\Pi' \supseteq \Pi$ may be consistent (see Example 6.11). Thus, for a set $E \subseteq H$ such that $T \cup E$ is incoherent, we cannot prune the supersets of E in 2^H to find an extension base of K. This fact makes the design and implementation of knowledge systems very difficult. To overcome this difficulty, we can use the characterization of all consistent answer sets of $T \cup E$ for any $E \subseteq H$ by analyzing the single program K^*. Therefore, another advantage of the transformational approach is that, in order to compute knowledge systems, we do not need any extra mechanism: any procedure based on the answer set (or stable model) semantics can be directly used for the transformed program.

We require three steps for the translation to K^*. To begin with, we will examine a knowledge system $K_0 = (T, H_0)$ such that H_0 is a set of rules without bodies (Section 6.4.1). The first transformation is performed from K_0 to a knowledge system $K_1 = (T, H_1)$ such that K_1 is not contradictory unless T is contradictory. Then, the

second translation constructs the target extended logic program K^* such that K^* is consistent under a certain condition. Next, in Section 6.4.2, we will describe the third translation from any knowledge system $K = (T, H)$ to the simple knowledge system in the form considered in Section 6.4.1, and then the three transformations will be finally combined.

In the following, for an extended logic program Π not containing integrity constraints (6.2), we denote the heads of the rules of Π as

$$\mathsf{Head}(\Pi) = \{\, L \mid (L \leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n) \in \Pi \,\}\,,$$

and the literals complementary to the heads of rules in Π as

$$\overline{\mathsf{Head}}(\Pi) = \{\, \overline{L} \mid (L \leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n) \in \Pi \,\}\,.$$

6.4.1 Simple Candidate Hypotheses

We firstly consider a simple knowledge system each of whose hypotheses is in the form of an assertion of a ground literal

$$L \leftarrow\ . \tag{6.8}$$

Since positive and negative literals are dealt with symmetrically in extended logic programs, we have no reason to restrict the simplest form of hypotheses to being positive. Let us consider a knowledge system $K_0 = (T, H_0)$ where H_0 is a set of rules of the form (6.8). We will translate K_0 to a non-contradictory program then to a consistent program.

Adding all literal assertions in H_0 to the program T would result in a contradictory or incoherent program. For example, when H_0 is contradictory, $T \cup H_0$ must be contradictory by Proposition 6.7. To *remove contradictions*, we can simply block the application of a hypothesis (6.8) if it happens that $\overline{L}$ is derived, by adding a formula $\mathit{not}\, \overline{L}$ to its body. Now let $K_1 = (T, H_1)$ be the knowledge system obtained from $K_0 = (T, H_0)$ by replacing each rule in H_0 of the form (6.8) with a rule in H_1 of the form

$$L \leftarrow \mathit{not}\, \overline{L}\,. \tag{6.9}$$

Then, $T \cup H_1$ is not contradictory unless T is contradictory. This is because H_1 does not contain a rule without *not* so that the following property can be shown to hold by Proposition 6.7.

Proposition 6.28 Let $K_0 = (T, H_0)$ and $K_1 = (T, H_1)$ be two knowledge systems as above. K_0 is contradictory if and only if $T \cup H_1$ is contradictory, which is then contradictory if and only if T is contradictory.

Notice that $T \cup H_1$ may be incoherent even if K_0 is consistent. Before we proceed further to remove incoherencies, let us consider the case that $T \cup H_1$ is consistent. Now, to characterize the extension bases of knowledge systems, we introduce the following definition.

Definition 6.29 Let Π and H be two extended logic programs. A consistent answer set S of Π is *H-maximal* if there is no consistent answer set S' of Π such that

$$S \cap \mathsf{Head}(H) \subset S' \cap \mathsf{Head}(H)\,.$$

For a knowledge system $K = (T, H)$, the distinction between the H-maximality of an answer set and an extension base of K is important. When $T \cup H$ is consistent, since no hypothesis is ignored, any answer set of $T \cup H$ is an extension base of K, but it may not be an H-maximal answer set of $T \cup H$. On the other hand, when S is an H-maximal answer set of $T \cup E$ for some set $E \subseteq H$, it may not be an extension base of K. In an H-maximal answer set, the hypotheses in a maximal subset of H are used in practice, whereas in an extension base, the hypotheses just take part in a maximal subset of H but some of them may be canceled so that their heads are not contained in the extension base.

Example 6.30 Suppose that we are given the knowledge system $K = (T, H)$ where $T = \{\, P \leftarrow \mathit{not}\, \neg P \,\}$ and $H = \{\, \neg P \leftarrow \mathit{not}\, P \,\}$. While both $S_1 = \{P\}$ and $S_2 = \{\neg P\}$ are extension bases of K (in fact, they are the answer sets of $T \cup H$), only S_2 is an H-maximal answer set of $T \cup H$.

For the first translation, the next proposition holds.

Proposition 6.31 Let $K_0 = (T, H_0)$ and $K_1 = (T, H_1)$ be two knowledge systems as above. Suppose that $T \cup H_1$ is consistent. If S is an H_1-maximal answer set of $T \cup H_1$, then S is an extension base of K_0.

Proof: We firstly prove that if S is an answer set of $T \cup H_1$, then S is an answer set of $T \cup {H_1}^S$. Suppose that S is an answer set of $T \cup H_1$. Since

$$\begin{aligned} {H_1}^S &= \{\, L \leftarrow \mid (L \leftarrow \mathit{not}\, \overline{L}) \in H_1,\ \overline{L} \notin S \,\} \\ &= \{\, (L \leftarrow) \in H_0 \mid \overline{L} \notin S \,\} \subseteq H_0 \end{aligned}$$

and $S = \alpha((T \cup H_1)^S) = \alpha((T \cup {H_1}^S)^S)$, S is an answer set of $T \cup {H_1}^S$.

Now, suppose that S is an H_1-maximal answer set of $T \cup H_1$. Suppose also to the contrary that S is not an extension base of K_0. Then, there exists a set E_0 $({H_1}^S \subset E_0 \subseteq H_0)$ such that $T \cup E_0$ is consistent. Let S' be an answer set of $T \cup E_0$. By ${H_1}^S \subset E_0 \subseteq {H_1}^{S'}$, clearly it holds that $S \cap \mathsf{Head}(H_1) \subset S' \cap \mathsf{Head}(H_1)$, contradicting the H_1-maximality of S. □

The converse of Proposition 6.31 does not hold: there is an extension base of $K_0 = (T, H_0)$ which is not an H_1-maximal answer set of $T \cup H_1$ (suppose a case that an extension base S contains neither L nor $\overline{L}$ for a literal $L \in \mathsf{Head}(H_1)$). Moreover, it cannot give every consistent answer set of $T \cup E_0$ for any set $E_0 \subseteq H_0$, which may be used for abduction.

Example 6.32 Let us consider the knowledge system $K_0 = (T, H_0)$ and its translated knowledge system $K_1 = (T, H_1)$, where

$$\begin{array}{rlll} T = \{ & \neg P \leftarrow \mathit{not}\, P\,, & & \\ & \leftarrow P, Q & \}\,, & \\ H_0 = \{ & P \leftarrow\,, & \quad H_1 = \{ & P \leftarrow \mathit{not}\, \neg P\,, \\ & Q \leftarrow & \}\,, & Q \leftarrow \mathit{not}\, \neg Q \quad \}\,. \end{array}$$

K_0 has two extension bases: $S_1 = \{ P \}$ and $S_2 = \{ \neg P, Q \}$. However, S_2 is the unique answer set of $T \cup H_1$. In S_1, neither Q nor $\neg Q$ holds. Note also that there is an answer set of $T = T \cup \emptyset$: $S_3 = \{ \neg P \}$, which cannot be obtained from the answer sets of $T \cup H_1$.

Another difficulty of Proposition 6.31 is that the consistency assumption for $T \cup H_1$ is indispensable. For example, we have seen in Section 6.1 and in Example 6.18 that an extended logic program with the CWA may be incoherent but the corresponding knowledge system may have extension bases. Therefore, adding all hypotheses $(L \leftarrow \mathit{not}\, \overline{L})$ in H_1 to the program T would result in an incoherent program even if T is consistent. Thus our next target is to *remove incoherencies*. In the following translation, we will characterize all consistent answer sets of $T \cup E_0$ for any set $E_0 \subseteq H_0$ as well as each extension base of K_0. The next lemma gives the background for the translation.

Lemma 6.33 Let T be a non-contradictory extended logic program, and E_1 a set of rules of the form (6.9). If $T \cup E_1$ has an answer set S, then for each rule in E_1 with the head L, S contains either L or $\overline{L}$ but not both of them.

> **Proof:** By Proposition 6.28, $T \cup E_1$ is not contradictory and hence S does not contain both L and $\overline{L}$. If S does not contain $\overline{L}$, then by $(L \leftarrow \mathit{not}\, \overline{L}) \in E_1$, $(T \cup E_1)^S$ contains $(L \leftarrow)$ and so $L \in \alpha((T \cup E_1)^S) = S$. If S does not contain L, then $\overline{L}$ must be contained in S because if $\overline{L} \notin S$ then $L \in S$ holds by the same argument as above contradicting $L \notin S$. $\Box$

The basic idea of the next translation is that we *expand* each incomplete extension base S of $K_0 = (T, H_0)$ by adding the extra literal $\overline{L}$ for each literal $L \in \mathsf{Head}(H_0)$

such that L is undefined in S. The augmented set of literals contains either L or $\overline{L}$ for each $L \in \mathsf{Head}(H_0)$ and is an answer set of $T \cup H_1 \cup H_2$ (H_2 is the added hypotheses) by Lemma 6.33.

Now, for each rule in H_1 of the form (6.9)

$$L \leftarrow not\, \overline{L}\,,$$

we shall consider the hypothesis of the opposite form

$$\overline{L} \leftarrow not\, L\,. \tag{6.10}$$

For H_1, we denote the set of opposite hypotheses of the form (6.10) as

$$\overline{H_1} = \{\, \overline{L} \leftarrow not\, L \mid L \in \mathsf{Head}(H_1) \,\}\,.$$

The result of the second translation is the extended logic program

$$K^* = T \cup H_1 \cup \overline{H_1}\,.$$

Furthermore, for knowledge systems $K_0 = (T, H_0)$ and $K_1 = (T, H_1)$, we shall impose the following restriction on the syntax of T (this restriction will be removed completely in Section 6.4.2).

> For any $(L \leftarrow) \in H_0$ (or $(L \leftarrow not\, \overline{L}) \in H_1$),
> every rule in T does not contain L in its head
> and contains neither $\overline{L}$ nor $not\, \overline{L}$ in its body. (6.11)

Example 6.34 Assume that the rule

$$\neg Ab(x) \leftarrow not\, Ab(x)$$

is a hypothesis in H. Then, the following rules satisfy the condition (6.11):

$$\begin{array}{l} Flies(x) \leftarrow Bird(x),\, \neg Ab(x)\,, \\ Ab(x) \leftarrow Ostrich(x)\,, \\ \neg Flies(x) \leftarrow not\, \neg Ab(x)\,, \end{array}$$

while the next three rules do not:

$$\begin{array}{l} Flies(x) \leftarrow Bird(x),\, not\, Ab(x)\,, \\ \neg Ab(x) \leftarrow Canary(x)\,, \\ \neg Flies(x) \leftarrow Ab(x)\,. \end{array}$$

The next is the main result of the second translation.

Theorem 6.35 Let $K_1 = (T, H_1)$ be a knowledge system such that H_1 is a set of rules of the form (6.9) and T satisfies the condition (6.11). If S is a consistent answer set of $T \cup E_1$ where E_1 is a subset of H_1, then

$$S' = S \cup \overline{\mathsf{Head}}(H_1 \setminus E_1) \tag{6.12}$$

is a consistent answer set of K^*. Moreover, every consistent answer set of K^* can be represented in the form (6.12) where S is a consistent answer set of $T \cup E_1$ for some set $E_1 \subseteq H_1$.

Proof: Let S be a consistent answer set of $T \cup E_1$ ($E_1 \subseteq H_1$). Since no literal $L \in \mathsf{Head}(H_1)$ appears in the head of any rule in T, for any literal $L \in \mathsf{Head}(H_1 \setminus E_1)$, it holds that $L \notin S$. Therefore, S' is consistent. By Lemma 6.33, $\mathsf{Head}({E_1}^S) \subseteq S$ and $(\overline{\mathsf{Head}}(E_1) \setminus \overline{\mathsf{Head}}({E_1}^S)) \subseteq S$, then it follows that $\mathsf{Head}({E_1}^S) \subseteq S'$, and

$$\begin{array}{ll} & \overline{\mathsf{Head}}(H_1 \setminus E_1) \cup (\overline{\mathsf{Head}}(E_1) \setminus \overline{\mathsf{Head}}({E_1}^S)) \\ = & (\overline{\mathsf{Head}}(H_1) \setminus \overline{\mathsf{Head}}({E_1}^S)) \subseteq S'. \end{array}$$

Since no rule in T contains $not\,\overline{L}$ for any $\overline{L} \in \overline{\mathsf{Head}}(H_1)$ in its body, it holds that $T^{S'} = T^S$ and

$${H_1}^{S'} = \{\, L \leftarrow \mid L \in \mathsf{Head}({E_1}^S) \,\} = {E_1}^S .$$

Now,

$$\begin{array}{ll} & \alpha((K^*)^{S'}) \\ = & \alpha(T^{S'} \cup {H_1}^{S'} \cup \overline{H_1}^{S'}) \\ = & \alpha(T^S \cup {E_1}^S \cup \{\, \overline{L} \leftarrow \mid L \in (\mathsf{Head}(H_1) \setminus \mathsf{Head}({E_1}^S)) \,\}) \\ = & \alpha((T \cup E_1)^S) \cup (\overline{\mathsf{Head}}(H_1) \setminus \overline{\mathsf{Head}}({E_1}^S)) \\ & \qquad \text{(since no rule in } T \text{ contains} \\ & \qquad\qquad \text{any } \overline{L} \in \overline{\mathsf{Head}}(H_1) \text{ in its body)} \\ = & S \cup \overline{\mathsf{Head}}(H_1 \setminus E_1) \quad (\text{by } (\overline{\mathsf{Head}}(E_1) \setminus \overline{\mathsf{Head}}({E_1}^S)) \subseteq S) \\ = & S' . \end{array}$$

Hence, S' is a consistent answer set of K^*.

To prove the second claim, take any answer set S' of K^*, and define

$$E_1 = \{\, (L \leftarrow not\,\overline{L}) \in H_1 \mid L \in S' \,\} .$$

Clearly, $E_1 \subseteq H_1$ and ${H_1}^{S'} = \{\, L \leftarrow \mid L \in \mathsf{Head}(E_1) \,\} = {E_1}^{S'}$. Then,

$$\begin{aligned} S' &= \alpha((T \cup H_1 \cup \overline{H_1})^{S'}) \\ &= \alpha(T^{S'} \cup {H_1}^{S'} \cup \overline{H_1}^{S'}) \\ &= \alpha(T^{S'} \cup {E_1}^{S'} \cup \{\, \overline{L} \leftarrow \mid L \in \mathsf{Head}(H_1 \setminus E_1) \,\}) \\ &= \alpha(T^{S'} \cup {E_1}^{S'}) \cup \overline{\mathsf{Head}}(H_1 \setminus E_1)\,. \end{aligned}$$

Now, let $S = \alpha(T^{S'} \cup {E_1}^{S'})$. Since $S' = S \cup \overline{\mathsf{Head}}(H_1 \setminus E_1)$, it holds that $T^{S'} = T^S$ and ${E_1}^{S'} = {E_1}^S$ by the condition (6.11). Therefore, $S = \alpha(T^S \cup {E_1}^S) = \alpha((T \cup E_1)^S)$. □

Example 6.36 Let us verify Theorem 6.35 in the example of the CWA introduced in Section 6.1. Let $K_1 = (T, H_1)$ be the knowledge system, where

$$\begin{aligned} T = \{\ & Q \leftarrow \neg P(A), \neg P(B)\,, \\ & \neg Q \leftarrow \qquad\qquad\quad \}\,, \\ H_1 = \{\ & \neg P(x) \leftarrow \mathit{not}\, P(x) \quad \}\,. \end{aligned}$$

In this case, $\overline{H_1} = \{\ P(x) \leftarrow \mathit{not}\, \neg P(x)\ \}$. There are three answer sets of $T \cup H_1 \cup \overline{H_1}$: ${S_1}' = \{\, \neg P(A), P(B), \neg Q \,\}$, ${S_2}' = \{\, P(A), \neg P(B), \neg Q \,\}$, and ${S_3}' = \{\, P(A), P(B), \neg Q \,\}$. By using the translation in the proof of the second claim of Theorem 6.35, we get the three corresponding answer sets:

$$\begin{aligned} &S_1 = \{\, \neg P(A), \neg Q \,\} \quad \text{for } T \cup \{\, \neg P(A) \leftarrow \mathit{not}\, P(A) \,\}\,, \\ &S_2 = \{\, \neg P(B), \neg Q \,\} \quad \text{for } T \cup \{\, \neg P(B) \leftarrow \mathit{not}\, P(B) \,\}\,, \text{ and} \\ &S_3 = \{\, \neg Q \,\} \quad \text{for } T \cup \emptyset = T\,. \end{aligned}$$

The next two properties characterize the knowledge system K_0 with simple hypotheses by the translated program K^*. These are the final results of this subsection.

Corollary 6.37 Let $K_0 = (T, H_0)$ be a knowledge system such that H_0 is a set of rules of the form (6.8) and T satisfies the condition (6.11). If S is a consistent answer set of $T \cup E_0$ for some $E_0 \subseteq H_0$, then

$$S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0) \tag{6.13}$$

is a consistent answer set of K^*. Moreover, every consistent answer set of K^* can be represented in the form (6.13) where S is a consistent answer set of $T \cup E_0$ for some set $E_0 \subseteq H_0$.

Proof: The first claim can be proved in a similar way to Theorem 6.35. To prove the second claim, for any answer set S' of K^*, we can define $E_0 = \{\, (L \leftarrow) \in H_0 \mid L \in S' \,\}$ and use the same argument as the previous proof. □

Theorem 6.38 Let $K_0 = (T, H_0)$ be the same knowledge system as in Corollary 6.37. If S is an extension base of K_0, then

$$S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0), \quad \text{where } E_0 = \{\, (L \leftarrow) \in H_0 \mid L \in S \,\}, \tag{6.14}$$

is an H_0-maximal answer set of K^*. Moreover, every H_0-maximal answer set of K^* can be represented in the form (6.14) where S is an extension base of K_0.

Proof: Suppose that S is an extension base of K_0. Then, S is an answer set of $T \cup E_0$ because $\mathsf{Head}(E_0) \subseteq S$. By Corollary 6.37, S' is a consistent answer set of K^*. Suppose to the contrary that S' is not an H_0-maximal answer set of K^*. Then, there is an answer set S'' of K^* such that $S' \cap \mathsf{Head}(H_0) \subset S'' \cap \mathsf{Head}(H_0)$. Since $E_0 = \{\, (L \leftarrow) \in H_0 \mid L \in S \,\} = \{\, (L \leftarrow) \in H_0 \mid L \in S' \,\}$, it holds that $E_0 \subset \{\, (L \leftarrow) \in H_0 \mid L \in S'' \,\}$. This contradicts the maximality of E_0 in 2^{H_0}. Hence, S' is an H_0-maximal answer set of K^*.

Now, we prove the second claim. Suppose that S' is an H_0-maximal answer set of K^*. By Corollary 6.37, S' can be represented by $S' = S \cup \overline{\mathsf{Head}}(H_0 \setminus E_0)$, where S is an answer set of $T \cup E_0$ and $E_0 = \{\, (L \leftarrow) \in H_0 \mid L \in S' \,\} = \{\, (L \leftarrow) \in H_0 \mid L \in S \,\}$. Suppose to the contrary that S is not an extension base of K_0. Then, there is a set F $(E_0 \subset F \subseteq H_0)$ such that $T \cup F$ is consistent. Let R be an answer set of $T \cup F$. By Corollary 6.37, $R' = R \cup \overline{\mathsf{Head}}(H \setminus F)$ is an answer set of K^*. By $E_0 \subset F$, clearly $\mathsf{Head}(E_0) \subset \mathsf{Head}(F) \subseteq R$. Therefore, $S' \cap \mathsf{Head}(H_0) \subset R \cap \mathsf{Head}(H_0) \subseteq R' \cap \mathsf{Head}(H_0)$. This contradicts the H_0-maximality of S'. □

Example 6.39 Let us consider the knowledge system $K_0 = (T, H_0)$, which is the same as Example 6.32, and the translated sets of hypotheses, H_1 and $\overline{H_1}$:

$$\begin{array}{rlrl} T = \{ & \neg P \leftarrow not\, P, & H_0 = \{ & P \leftarrow, \\ & \leftarrow P, Q \quad \}, & & Q \leftarrow \quad \}, \\ H_1 = \{ & P \leftarrow not\, \neg P, & \overline{H_1} = \{ & \neg P \leftarrow not\, P, \\ & Q \leftarrow not\, \neg Q \quad \}, & & \neg Q \leftarrow not\, Q \quad \}. \end{array}$$

There are three answer sets of $K^* = T \cup H \cup \overline{H}$: ${S_1}' = \{\, P, \neg Q \,\}$, ${S_2}' = \{\, \neg P, Q \,\}$, and ${S_3}' = \{\, \neg P, \neg Q \,\}$. Of these, ${S_1}'$ and ${S_2}'$ are two H_0-maximal answer sets of K^*, and they correspond to the expansions of the two extension bases of K_0: $S_1 = \{\, P \,\}$ and $S_2 = \{\, \neg P, Q \,\}$. Note that ${S_3}'$ is the expansion of the answer set of T: $S_3 = \{\, \neg P \,\}$.

6.4.2 Complex Candidate Hypotheses

In the last subsection, we considered a knowledge system $K = (T, H)$ where H is restricted to being either a set of rules of the form (6.8) or a set of rules of the form (6.9). Moreover, we considered only the case where a set of rules T satisfies the condition (6.11). In this subsection, we remove all of these restrictions: we allow *any* extended logic program for both T and H.

Example 6.40 Let us firstly consider the case in which T does not satisfy the condition (6.11) for $K = (T, H)$, where H is a set of hypotheses of the form (6.9). Suppose that K is the following knowledge system:

$$\begin{array}{rl} T = \{ & Q \leftarrow P\,, \\ & Q \leftarrow \neg P\,, \\ & \neg Q \leftarrow \qquad \}\,, \\ H = \{ & \neg P \leftarrow not\, P \quad \}\,. \end{array}$$

K does not satisfy the condition (6.11) because P appears in the body of the first rule of T. It is easy to see that K has the unique extension base: $S = \{ \neg Q \}$, which is the only answer set of T. However, when we introduce the opposite hypothesis, $\overline{H} = \{ P \leftarrow not\, \neg P \}$, we see that the program $T \cup H \cup \overline{H}$ is incoherent. Thus Theorem 6.35 cannot be used in this case. This is because neither P nor $\neg P$ can be consistently added to T, but introducing $\overline{H}$ forces the answer set to include either of them by Lemma 6.33.

We shall translate any knowledge system K to an extended logic program K^*. The basic idea is "naming hypotheses" and is similar to Poole [1988]. After the translation, we can use the results for simple hypotheses presented in the last subsection.

Now, let $K = (T, H)$ be any knowledge system. For each rule $C \in H$, we shall associate a propositional symbol δ_C which does not appear elsewhere in K. [8] For any subset E of H, we define the following sets of rules:

$$\begin{array}{rcl} \Delta_0(E) & = & \{ \delta_C \leftarrow \mid C \in E \}\,, \\ \Delta(E) & = & \{ \delta_C \leftarrow not\, \neg\delta_C \mid C \in E \}\,, \\ \overline{\Delta}(E) & = & \{ \neg\delta_C \leftarrow not\, \delta_C \mid C \in E \}\,, \quad \text{and} \\ \Gamma(E) & = & \{ L \leftarrow \delta_C, L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \mid \\ & & \qquad C = (L \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n) \in E \} \\ & & \cup \{ \leftarrow \delta_C, L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n \mid \\ & & \qquad C = (\leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n) \in E \}\,. \end{array}$$

[8] If a hypothesis C contains n distinct free variables $\mathbf{x} = x_1, \ldots, x_n$, then we can name C with $\delta_C(\mathbf{x})$ where δ_C is an n-ary predicate symbol appearing nowhere in K. Note that every variable appearing in a rule is a free variable in our language.

For $K = (T, H)$, we define the extended logic program K^* as:

$$K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H)\,.$$

Before analyzing the program K^*, let us first consider a knowledge system

$$K_0 = (T \cup \Gamma(H), \Delta_0(H))\,.$$

This knowledge system has only atomic form of hypotheses and satisfies the condition (6.11) because no $\delta_C \in \mathsf{Head}(\Delta_0(H))$ appears in any rule other than in the body of one rule in $\Gamma(H)$. [9] Therefore, we can apply Corollary 6.37 and Theorem 6.38 for K_0.

The basic property of the translation is shown by the next theorem.

Theorem 6.41 Let $K = (T, H)$ be any knowledge system, and E a subset of H. S is a consistent answer set of $T \cup E$ if and only if

$$S' = S \cup \mathsf{Head}(\Delta_0(E))$$

is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$.

Proof: Suppose that S is an answer set of $T \cup E$. Then S' is obviously consistent. It is easy to see that the knowledge system $(T \cup \Gamma(E), \Delta_0(E))$ satisfies the condition (6.11). Therefore, $T^{S'} = T^S$ because S' does not contain any new literal other than the names of hypotheses of E. Similarly, $\Gamma(E)^{S'} = \Gamma(E)^S$, and $\Delta_0(E)^{S'} = \Delta_0(E) = \{\, \delta_C \leftarrow \mid C \in E \,\}$. Now,

$$\begin{aligned}
& \alpha((T \cup \Gamma(E) \cup \Delta_0(E))^{S'}) \\
= \;& \alpha(T^S \cup \Gamma(E)^S \cup \Delta_0(E)) \\
= \;& \alpha(\, T^S \cup \{\, \delta_C \leftarrow \mid C \in E \,\} \cup \{\, L \leftarrow L_1, \ldots, L_m \mid \\
& \qquad (L \leftarrow L_1, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n) \in E\,, \\
& \qquad\quad L_{m+1}, \ldots, L_n \notin S' \,\}\,) \\
& \quad \text{(by unfolding the rules of } \Gamma(E)^S \text{ by } \Delta_0(E)) \\
= \;& \alpha(T^S \cup E^S \cup \Delta_0(E)) \\
= \;& \alpha((T \cup E)^S) \cup \mathsf{Head}(\Delta_0(E)) \\
= \;& S \cup \mathsf{Head}(\Delta_0(E)) \quad (\text{by } S \cap \mathsf{Head}(\Delta_0(E)) = \emptyset) \\
= \;& S'\,.
\end{aligned}$$

Hence, S' is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$.

[9] We can allow T to include rules containing $\delta_C \in \mathsf{Head}(\Delta_0(H))$ within the restriction (6.11). Since these rules are not necessary for our purpose, we do not pursue this possibility further (see also Example 6.46).

On the other hand, suppose that S' is a consistent answer set of $T \cup \Gamma(E) \cup \Delta_0(E)$. Since $S \cap \mathsf{Head}(\Delta_0(E)) = \emptyset$, we can immediately identify S from S'. By using the same translation as above, we see that

$$\begin{aligned} S' &= \alpha(T^{S'} \cup \Gamma(E)^{S'} \cup \Delta_0(E)^{S'}) \\ &= \alpha(T^S \cup E^S \cup \Delta_0(E)) \\ &= \alpha((T \cup E)^S) \cup \mathsf{Head}(\Delta_0(E)) \,. \end{aligned}$$

Since $\alpha((T \cup E)^S) \cap \mathsf{Head}(\Delta_0(E)) = \emptyset$, it holds that $S = \alpha((T \cup E)^S)$. Hence, S is an answer set of $T \cup E$. □

By combining Theorem 6.41 and Corollary 6.37, we get the following result. Every answer set of any consistent theory from $K = (T, H)$ can be characterized by an answer set of $K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H)$, and vice versa.

Corollary 6.42 Let $K = (T, H)$ be any knowledge system. If S is a consistent answer set of $T \cup E$ where E is a subset of H, then

$$S' = S \cup \mathsf{Head}(\Delta_0(E)) \cup \overline{\mathsf{Head}}(\Delta_0(H \setminus E)) \tag{6.15}$$

is a consistent answer set of K^*. Moreover, every consistent answer set S' of K^* can be represented in the form (6.15) where S is a consistent answer set of $T \cup E$ for some set $E \subseteq H$.

Corollary 6.42 shows that for (T, H) if $T \cup E$ $(E \subseteq H)$ has a consistent answer set S then δ_C can be consistently added to S for every hypothesis C in E and the negated names of all other hypotheses can be also added to S, and that we can find these answer sets of the consistent theories from (T, H) by removing all of positive and negative naming hypotheses from the answer sets of K^*.

Finally, we can characterize the extension bases of $K = (T, H)$ by combining Theorem 6.38, Theorem 6.41 and Corollary 6.42.

Corollary 6.43 Let $K = (T, H)$ be any knowledge system. If S is an extension base of K, that is, an answer set of $T \cup E$ for a maximal subset E of H such that $T \cup E$ is consistent, then

$$S' = S \cup \mathsf{Head}(\Delta_0(E)) \cup \overline{\mathsf{Head}}(\Delta_0(H \setminus E))$$

is a $\Delta_0(H)$-maximal answer set of K^*.

Conversely, if S' is a $\Delta_0(H)$-maximal answer set of K^*, then

$$S = S' \setminus (\mathsf{Head}(\Delta_0(H)) \cup \overline{\mathsf{Head}}(\Delta_0(H)))$$

is an extension base of K.

Example 6.44 Let $K = (T, H)$ be the knowledge system introduced in Example 6.40:

$$\begin{array}{rl} T = \{ & Q \leftarrow P\,, \\ & Q \leftarrow \neg P\,, \\ & \neg Q \leftarrow \quad \}\,, \\ H = \{ & \neg P \leftarrow not\, P \quad \}\,. \end{array}$$

Now, we can name the hypothesis as follows.

$$\begin{array}{rcl} \Delta_0(H) & = & \{\ \delta_{\neg P \leftarrow not\, P} \leftarrow \}\,, \\ \Gamma(H) & = & \{\ \neg P \leftarrow \delta_{\neg P \leftarrow not\, P},\ not\, P\ \}\,. \end{array}$$

Recall that K has the unique extension base: $S = \{\ \neg Q\ \}$. It is easy to check that $S' = S \cup \{\ \neg\delta_{\neg P \leftarrow not\, P}\ \}$ is the unique answer set of $K^* = T \cup \Gamma(H) \cup \Delta(H) \cup \overline{\Delta}(H)$.

Example 6.45 Let us see how we can restore the consistency of an incoherent program Π. We construct the knowledge system $(\emptyset, \Pi)$ according to Section 6.3.2 and apply the reduction techniques. For example, consider the knowledge system $K = (\emptyset, \Pi)$, where

$$\Pi = \{\ P \leftarrow not\, P\ \}\,.$$

In this case, $S = \emptyset$ is the unique extension base of K. Now,

$$\begin{array}{rcl} \Delta_0(\Pi) & = & \{\ \delta_{P \leftarrow not\, P} \leftarrow \} \\ \Gamma(\Pi) & = & \{\ P \leftarrow \delta_{P \leftarrow not\, P},\ not\, P\ \} \end{array}$$

The unique answer set of K^* is

$$S' = S \cup \{\ \neg\delta_{P \leftarrow not\, P}\ \}\,.$$

Example 6.46 Consider the knowledge system $K = (T, H)$ introduced in Example 6.13:

$$\begin{array}{rl} T = \{ & \neg Flies(x) \leftarrow Penguin(x)\,, \\ & Bird(x) \leftarrow Penguin(x)\,, \\ & Bird(Polly) \leftarrow\,, \\ & Penguin(Tweety) \leftarrow \quad \}\,, \\ H = \{ & Flies(x) \leftarrow Bird(x) \quad \}\,. \end{array}$$

In this case, we can name the hypothesis as follows.

$$\begin{array}{rcl} \Delta_0(H) & = & \{\ Birdflies(x) \leftarrow \}\,, \\ \Gamma(H) & = & \{\ Flies(x) \leftarrow Birdflies(x),\ Bird(x)\ \}\,. \end{array}$$

Then, we see that there is the unique $\Delta_0(H)$-maximal answer set of K^*:

$$S' = \{\, Bird(Polly),\, Penguin(Tweety),\, Bird(Tweety),\, Flies(Polly),\\ Birdflies(Polly),\, \neg Flies(Tweety),\, \neg Birdflies(Tweety) \,\}\,.$$

Removing all the naming literals from S', we get the unique extension base S of K:

$$S = \{\, Bird(Polly),\, Penguin(Tweety),\\ Bird(Tweety),\, Flies(Polly),\, \neg Flies(Tweety) \,\}\,.$$

The difference between Poole's system and ours with respect to the naming is that the naming in [Poole, 1988] has the effects of introducing normal default rules, for example,

$$\frac{:\, \mathsf{M}\, Bird(x) \supset Flies(x)}{Bird(x) \supset Flies(x)}\,.$$

where $\supset$ is classical implication. This default rule causes unintended side effects: from $\neg Flies(Sam)$ we can conclude $\neg Bird(Sam)$ (this should not be allowed because we do not know the reason for Sam's inability to fly; Sam might be a penguin). To prevent such an inference, we must add a fact like $(\neg Flies(x) \supset \neg Birdflies(x))$. But, this clause further causes another side effects: from the hypothesis $Birdflies(Paul)$ and the contrapositive of $(\neg Flies(x) \supset \neg Birdflies(x))$, we can conclude $Flies(Paul)$. To prevent the last inference, we must use this fact as a *constraint*. In our case, both kinds of pruning rules are unnecessary.

6.5 Reduction to General Logic Programs

The question now is how to compute the proposed framework for theory formation. Since we have seen that every knowledge system can be transformed to a single extended logic program, we can use methods to compute answer sets of extended logic programs. For this purpose, Gelfond and Lifschitz [1990] show how to reduce an extended logic program to a general logic program. The method is to replace every classical negation in a program Π with a new propositional symbol, for example, $\neg A$ is replaced by A'. We call the resulting general logic program (denoted as Π^+) the *positive form* of Π. However, even if the original extended logic program is incoherent, its positive form may have stable models.

Example 6.47 Let Π be the extended logic program shown in the example of the CWA in Section 6.1, and Π^+ its positive form:

$$\Pi = \{\quad Q \leftarrow \neg P(A), \neg P(B)\,, \qquad\qquad \Pi^+ = \{\quad Q \leftarrow P(A)', P(B)'\,,$$
$$\neg Q \leftarrow\,, \qquad\qquad\qquad\qquad\qquad\quad Q' \leftarrow\,,$$
$$\neg P(x) \leftarrow not\, P(x) \qquad \}\,, \qquad\qquad\qquad\quad P(x)' \leftarrow not\, P(x) \qquad \}\,.$$

While Π is incoherent, Π^+ has a stable model $\{ P(A)', P(B)', Q, Q' \}$.

Note that not every incoherent program may be translated to a general logic program that has inconsistent stable models; it may remain incoherent (for example, $\Pi = \{ P \leftarrow not\, P \}$). Conversely, not every translated general program that has inconsistent stable models may correspond to an incoherent extended logic program; it may be translated from a contradictory program (for example, $\Pi^+ = \{ P \leftarrow,\ P' \leftarrow \}$). Anyway, we need a mechanism to check whether the resulting stable models have a pair of complementary propositions, say A and A'. To discard each stable model possessing a pair, we can represent pruning rules as *integrity constraints* (i.e., rules with no heads). For example, for each atom A such that both positive and negative literals appear in the program, we may add an extra integrity constraint:

$$\leftarrow A, A' . \tag{6.16}$$

Proposition 6.48 Let Π be any extended logic program, Π_P the set of rules not containing *not* in Π, and Π^{+^*} ($\Pi_P^{+^*}$) the general logic program consisting of all rules of the positive form of Π (Π_P) together with all integrity constraints (6.16) for all ground atoms A.
(1) Π is consistent if and only if Π^{+^*} is consistent. Furthermore, every stable model S' of Π^+ corresponds to an answer set S of Π, and vice versa.
(2) Π is contradictory if and only if $\Pi_P^{+^*}$ is contradictory.
(3) Π is incoherent if and only if $\Pi_P^{+^*}$ is not contradictory and Π^{+^*} is not consistent.

> **Proof:** (1) is a direct consequence of [Gelfond and Lifschitz, 1990, Proposition 2], which shows that Π is consistent if and only if Π^+ has a consistent stable model. For the proof of (2) and (3), we can identify whether an inconsistent program Π is contradictory or incoherent by Proposition 6.7, and the results follow. □

By using the above proposition, we can use any proof procedure for computing hypothetical reasoning proposed in this chapter. An abductive proof procedure for the stable model semantics of general logic programs is proposed by Eshghi and Kowalski [1989] for call-consistent programs. Procedures to compute stable models satisfying integrity constraints have been proposed by Satoh and Iwayama [1991] using a TMS-like bottom-up manner, and by Inoue, Koshimura and Hasegawa [1992a] using a model-generation theorem prover.

Alternative methods to compute the theory formation framework can be conceived. Since we have seen that every rule in H of a knowledge system $K = (T, H)$ can be transformed to the unique naming hypothesis, we can use *nonmonotonic ATMSs* [Dressler, 1989; Junker, 1989] to compute explanations of each atom. The consistency maintenance is then performed by pruning each set of hypotheses subsumed by a minimal *nogood*.

6.6 Discussion

In this section, we compare the proposed framework to other hypothetical reasoning systems based on logic programming. Especially, here we see how a lot of previous proposals for default and abductive reasoning as well as contradiction removals in the field of logic programming can be explained uniformly within the framework of knowledge systems. Not only is a unified view given, but each individual proposal is generalized. Furthermore, some problems in previous proposals can be solved by using our theory formation framework.

6.6.1 Abductive Logic Programming

There are some proposals for abduction by using logic programming.

Eshghi and Kowalski [1989] define an abductive framework that can be written as a knowledge system (T, H), where T is a definite Horn logic program (with integrity constraints) and H is a set of simple candidate hypotheses (called *abducibles*) of the form $A \leftarrow$ where A is a ground atom. This framework is used to give an abductive interpretation of negation as failure in general logic programs within the stable model semantics. Their framework is expanded by Kakas and Mancarella [1990] who define an abductive framework (T, H) where T is a general logic program (with integrity constraints) and H is a set of abducibles. Kakas and Mancarella define a *generalized stable model* of (T, H) as any consistent stable model of $T \cup E$ where E is any subset of H. Thus, our knowledge system for abduction is a generalization of these abductive frameworks because ours allows any extended logic programs for both facts and candidate hypotheses.

In addition to [Kakas and Mancarella, 1990], Gelfond [1990] proposes another abductive framework for logic programs. Gelfond allows the facts T to be an *extended disjunctive database* that is a set of rules of the form:

$$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n$$

where $n \geq m \geq l \geq 0$ and each L_i is a literal. Here, the symbol "|" means a connective of disjunctions, whose semantics is given in [Gelfond and Lifschitz, 1991]. Gelfond considers the hypotheses H of the form of literal assertions (6.8). It is possible to extend our framework by allowing such programs for both facts and hypotheses according to the semantics. Note that the definitions of *explanations* are different between [Kakas and Mancarella, 1990] and [Gelfond, 1990]: the former uses $explains_1$, whereas the latter uses $explains_2$ (see Section 6.3.4).

It should be also noted that when some candidate hypotheses represent default knowledge, abductive frameworks by [Gelfond, 1990; Kakas and Mancarella, 1990]

cannot be applied to default reasoning directly, unlike Poole's [1988] framework. As explained in Section 6.3.4, the fact that a formula has an explanation does not imply that the formula is true in an extension base of the knowledge system.

6.6.2 Exceptions

Kowalski and Sadri [1990] propose a technique for contradiction removals within the context of default reasoning. Their method deals with a simple form of *exceptions*, and translates each rule representing a prototypic property into a normal default rule. For instance, in Example 6.13, the extended logic program

$$T \cup \{ Flies(x) \leftarrow Bird(x), not \neg Flies(x) \}$$

is obtained from $T \cup H$ by using the Kowalski-Sadri transformation, and has the unique answer set that is identical to the extension base of K. The reason why their translation works is that the exceptional rule

$$\neg Flies(Tweety) \leftarrow Penguin(Tweety)$$

cancels the normal default rule

$$Flies(Tweety) \leftarrow Bird(Tweety), not \neg Flies(Tweety).$$

Comparing with our method, there are two limitations in Kowalski and Sadri's technique.

First, in their method, rules are automatically divided into two (those having positive literals as heads and those having negative literals as heads) so that negative literals are always exceptions of the positive ones with the same predicates. Kowalski and Sadri, however, claim that their technique can be extended to deal with exceptions with individual rules rather than entire rules and with exceptions having positive conclusions. But if we allow these mixed exceptions at the same time, then we have to take care of the semantics for each exception individually because the original answer set semantics of [Gelfond and Lifschitz, 1990] is changed in [Kowalski and Sadri, 1990]. Therefore, the techniques proposed in this chapter are more flexible.

Example 6.49 If the program Π consists of the following two contradictory rules

$$P \leftarrow ,$$
$$\neg P \leftarrow ,$$

then by the Kowalski-Sadri transformation, the first rule is replaced by

$$P \leftarrow not \neg P ,$$

and we obtain $\{\neg P\}$ as the unique answer set. If we prefer the positive conclusion instead, then we will obtain $\{P\}$. Thus, we cannot restore consistency without determining which literal we should regard as the exception. On the other hand, if we consider the knowledge system $(\emptyset, \Pi)$, we can obtain two alternative extension bases $\{P\}$ and $\{\neg P\}$, without committing to an explicit choice of exceptions.

Second, our framework for default reasoning and contradiction removals can be applied to much broader classes of default knowledge than their method because any extended logic program can be a set of candidate hypotheses. Especially, our notion of extension bases is not restricted to dealing only with exceptions, but can be applied to other types of default reasoning where exceptions may not be given explicitly for default knowledge. For instance, Examples 6.15 and 6.18 cannot be dealt with by [Kowalski and Sadri, 1990]. In Example 6.15, there is no exception of the hypothesis

$$Shaves(Noel, x) \leftarrow not\, Shaves(x, x)\,,$$

for $\neg Shaves(Casanova, Casanova)$ is not an exception. Thus, the hypothesis cannot be translated in the same way as Example 6.13. In case of the CWA, since hypotheses are inherently expressed by normal default rules, the Kowalski-Sadri translation does not change the meaning of the original program that may cause incoherency.

6.6.3 Truth Maintenance

We can compare our method for contradiction removals (Section 6.3.2) to the TMS-style computation. According to Elkan [1990], a set of rules for Doyle's [1979] TMS can be identified with a propositional general logic program with integrity constraints, and the TMS computes a consistent stable model of the program in a bottom-up manner. Classical negation is not incorporated in the TMS.

However, inconsistency resolution in our framework is different from the TMS-style contradiction removals in [Doyle, 1979; Giordano and Martelli, 1990; Elkan, 1990]. When a contradiction occurs, the TMS imposes a new rule in order to believe a literal that has not been believed. As a result of contradiction removals, the TMS may fail to output a stable model of the original program. Elkan [1990] claims that when the TMS finds an inconsistent stable model, it should choose another stable model of the program, if one exists. However, such a strategy is not tolerant of incoherent programs because it does not output anything if the program has no consistent stable model. Giordano and Martelli [1990] consider all possible models that may be output by TMS's contradiction removals (called *dependency-directed backtracking*). Although their method reflects an incremental use of the TMS, its model theory is no longer stable model semantics in the sense that contrapositives

of the original rules are interpreted to be valid and that literals interpreted to be false by negation as failure in the original program can be believed through those contrapositive rules. On the other hand, since our method for inconsistency resolution deals with retractable hypotheses explicitly, hypotheses alone are invalidated and other rules are never changed. In this sense, our knowledge system can be considered as a generalization of nonmonotonic ATMSs [Dressler, 1989; Junker, 1989], which handle TMS rules together with abducibles.

Besides the TMSs, there are some proposals for inconsistency resolution in nonmonotonic reasoning. For autoepistemic logic [Moore, 1985], Morris [1989] defines *stable closures* when there is no stable expansion. His proposal is motivated by dependency-directed backtracking in the TMS, and therefore some formulas are believed to remove inconsistencies. Again, we do not add any new formulas but remove a minimal set of hypotheses. For default logic, Guerreiro, Casanova and Hemerly [1990] propose an alternative definition of extensions. Although this definition is quite different from ours, the underlying idea is similar because default rules are allowed to be ignored in their extensions to keep the consistency but no default rule can be dispensed with unless it is necessary to do so. We consider such defeasible default rules only for some distinguished rules, rather than for all the rules.

6.6.4 Pereira et al.

Recent work by Pereira, Aparício and Alferes [1991a] independently concerns how to represent knowledge for default reasoning by using extended logic programs. By their framework, priorities among default knowledge can be formulated within a single program. Therefore, their goal is similar to our proposals described in Section 6.4. The differences between [Pereira *et al.*, 1991a] and ours are as follows.

First, [Pereira *et al.*, 1991a] uses the well founded semantics [van Gelder *et al.*, 1991] as well as the three-valued stable model semantics [Przymusinski, 1990a] by permitting the use of classical negation. As we have already argued, the theory formation framework itself is independent of the underlying semantics. However, some remarks should be made here. The well founded semantics provides "cautious" (or skeptical) conclusions rather than alternative solutions. On the other hand, the answer set semantics on which our framework is based is very suitable for determining what holds in each of the single extensions. In particular, viewing each extension as a theory is indispensable for abduction [Kowalski, 1990].

Second, a central concern of [Pereira *et al.*, 1991a] is to "hack" programs so as to deal with finer issues in default reasoning. We define the framework meta-theoretically just by separating the knowledge into two: concrete knowledge and hypothetical knowledge. Using H-maximality for $K = (T, H)$ enables us to represent default

knowledge in a more concise and understandable form. Our translation is just to show that our theory formation framework can be represented in a single program for computation, and the resulting program is not intended at all to be the user-provided representation of knowledge.

Third, they use both naming hypotheses (which we described in Section 6.4.2) and the cancellation technique similar to the Kowalski-Sadri transformation [Kowalski and Sadri, 1990] (adding $not\,\overline{L}$ to the body of a rule whose head is L). This is more complicated than our translation because we require only the naming technique. As we have already remarked in Section 6.6.2, it turns out that the cancellation technique is less useful than naming hypotheses because the cancellation does not change rules if they represent the CWA and because exceptions have to be explicitly listed for the cancellation to work. We can thus deal easily with contradiction removals and abduction as well as default reasoning within the unified framework for theory formation.

In fact, Pereira et al. also show an approach to contradiction removals within the well-founded semantics in a separate paper [Pereira *et al.*, 1991b],[10] but this is based on an entirely different technique from [Pereira *et al.*, 1991a]. Moreover, this method is applicable only for a special kind of extended logic programs whose inconsistencies can be removed by retracting some negation-as-failure formulas from rules in the programs. Thus, Example 6.49 (where no *not* appears in the program) cannot be handled by their method.

6.6.5 Strong Introspection

Gelfond [1991] also considers a flexible representation of a broad class of incomplete information. In [Gelfond, 1991], he extends the syntax and semantics of extended disjunctive databases [Gelfond and Lifschitz, 1991] by a new modal operator (called *strongly introspective*), and develops a theory to specify negative information from extended disjunctive databases. Like our argument in Section 6.3.3, Gelfond claims that the simple inclusion of the (Reiter's) CWA rule $\neg A \leftarrow not\,A$ for each ground atom A is not suitable in general, and suggests a new CWA rule that infers a negative literal $\neg A$ if and only if no answer set of the program contains A. The differences between Gelfond's approach and ours are as follows.

Since disjunctive information causes multiple answer sets, Gelfond uses a new modal operator $\mathbf{M}$ for his CWA rule: $\neg P$ is true in a belief set if $\mathbf{M}P$ is not true, where an atom $\mathbf{M}P$ means that P appears in some answer set. On the other hand, our CWA is considered for a non-disjunctive program Π, and a maximal subset of

[10] Note that, unlike for general logic programs, the well-founded semantics extended with classical negation may lead to incoherency (that is, no models) for extended logic programs.

Reiter's CWA rules CW is considered for each answer set of Π: $\neg P$ is inferred if P is not true in every extension base of (Π, CW). These two specifications do not coincide if our CWA is applied to extended disjunctive databases.

Example 6.50 Let us consider the following extended disjunctive database:

$$\begin{aligned} P \mid Q &\leftarrow \ , \\ R &\leftarrow \neg P \, , \\ R &\leftarrow \neg Q \, . \end{aligned}$$

If we add Reiter's CWA rules for P and Q:

$$\begin{aligned} \neg P &\leftarrow not\, P \, , \\ \neg Q &\leftarrow not\, Q \, , \end{aligned}$$

then we get two augmented answer sets, $\{P, \neg Q, R\}$ and $\{\neg P, Q, R\}$. Hence, we can derive R under our CWA. But, if we use Gelfond's CWA rules for P and Q:

$$\begin{aligned} \neg P &\leftarrow not\, \mathbf{M}P \, , \\ \neg Q &\leftarrow not\, \mathbf{M}Q \, , \end{aligned}$$

we get two belief sets, $\{P\}$ and $\{Q\}$, and thus R cannot be derived by his specification.

In some cases, Gelfond's CWA rules are more appropriate for specifying the closed world assumption for extended disjunctive databases. In general, we would like to choose freely either form of CWA rule for each predicate. An extension of our knowledge system by incorporating this strong introspection is an important subject for future investigation.

Another, but more important difference appears when we deal with a *consistent program whose augmentation is inconsistent*. For example, for the extended logic program

$$\Pi = \{ \ \leftarrow \neg P \ \} \, ,$$

Gelfond's CWA rule for P adds $\neg P$ so that the augmentation becomes inconsistent. However, the knowledge system

$$K = (\Pi, \ \{ \neg P \leftarrow not\, P \})$$

does not allow the use of Reiter's CWA rule for P. Thus, our main concern is to keep the consistency of augmentations since we add a maximally consistent set of negative literals to each answer set. This problem is not considered by Gelfond's specification.

6.6.6 Priority

The last question is how to divide our commonsense knowledge into the factual (or background) program and the candidate hypotheses. In other words, how should each rule in an extended logic program be placed in either T or H for a knowledge system $K = (T, H)$? One easy way is to associate *integrity constraints* (see Section 6.2.2) with T and other rules with H. Then, integrity constraints must be satisfied by every extension base but a minimal set of other rules may be disregarded to keep the consistency. This view of theory formation with integrity constraints was first introduced by Goebel, Furukawa and Poole [1986]. In their Theorist-S system, each rule in H is a definite clause (without negation as failure) and integrity constraints in F are used to retract a minimal set of hypotheses from H until the consistency with F is restored. The theories obtained are exactly our extension bases.

The notion of maximally consistent combination of logic programs is also employed by Baral, Kraus and Minker [1991]. They combine multiple programs, say Π_1 and Π_2, which are mutually inconsistent with respect to integrity constraints IC, into a single maximally consistent program Π'. Their goal can be simulated by considering the knowledge system $K = (IC, \Pi_1 \cup \Pi_2)$, then by translating it into a single logic program K^*. In [Baral *et al.*, 1991], the input programs are restricted to stratified logic programs without classical negation, and the output is a disjunctive stratified program. In contrast, our input programs can be extended logic programs, not necessarily stratified, and the output is an extended logic program too, without introducing disjunctions.

A more general view of knowledge systems is to divide the program into subprograms (*categories*) in accordance with the degrees of credibility of default knowledge, where the priority is determined depending on the problem domain. This view of hypothetical reasoning is exactly the same idea as Rescher [1964]. When there are more than two categories for a problem, we may have an extended knowledge system like $K = (H_0, H_1, \ldots, H_n)$. Along this line, an extended framework for hypothetical reasoning based on first-order logic is considered by Brewka [1989b] as an extension of Poole's [1988] framework. It may be possible to extend our framework in the same way as [Brewka, 1989b].

Finally, hypothetical reasoning can be used in conjunction with an inductive system. [11] An inductive method can provide estimates of the correctness or the generality of knowledge, and can serve the basis for deciding whether each rule should be placed in T, H_0 or $H_1, \ldots, H_n$. An extended knowledge system then could be used to identify potential inconsistencies and to suggest how to remove them. This last information will be used for further refinements in the inductive system. In this way,

[11] This view of knowledge systems was suggested by Pavel Brazdil.

we can represent and classify knowledge at the same time that we can use knowledge systems for commonsense reasoning.

6.7 Summary

By expanding the notion of classical negation and the consistency, we proposed a uniform framework based on theory formation called knowledge systems, which can be used to represent various forms of reasoning of interest in logic programming. In a knowledge system, candidate hypotheses H are dealt with as a part of knowledge distinct from a program T about the world, and hypotheses are used to augment the theory and to predict what we expect to be true.

One of the main tasks of a knowledge system is to find a maximal subset E of H such that there is a consistent answer set of the extended logic program $T \cup E$. If adding hypotheses causes inconsistencies, then a minimal set of hypotheses can be ignored to remove inconsistencies. This framework can also be used for abduction, and for important functions of logic programming such as contradiction removals and the closed world assumptions.

We also proposed the translation of a knowledge system K to a single extended logic program K^* such that each answer set of K^* exactly corresponds to an answer set of a consistent theory from K, and vice versa. By using this translation, we can use any query-answering or model-generation procedure to compute hypothetical reasoning.

Although theory formation and hypothetical reasoning have mainly been proposed in AI using first-order logic, we suggested that these concepts are also very useful for logic programs with nonmonotonic behaviors. As a result, we can conclude that hypothetical reasoning is not only essential for many reasoning aspects of AI, but very helpful for a uniform approach to solving many important problems for logic programming.

Chapter 7

Bottom-Up Computation of Logic Programs

In this chapter, we are concerned with a new algorithm for computing *answer sets* of every class of (function-free) logic programs and deductive databases containing both negation as failure and classical negation. The proposal is based on bottom-up, incremental, backtrack-free computation of the minimal models of positive disjunctive programs, together with integrity constraints over beliefs and disbeliefs. Our translation method not only provides a simple fixpoint characterization of answer sets, but also is very helpful for understanding under what conditions each model is "stable" or "unstable". The procedure has been implemented on top of the model generation theorem prover MGTP on a parallel inference machine, and has been applied to a legal reasoning system.

7.1 Introduction

This chapter presents a novel and simple procedure which computes the models of logic programs containing *negation-as-failure* formulas. In traditional top-down proof procedures such as SLDNF-resolution [Clark, 1978; Lloyd, 1984], *not* P succeeds if there is no top-down proof of P; the meaning of negation as failure is only procedural. On the other hand, in recent theories of logic programming and deductive databases, declarative semantics have been given to extensions of logic programs, where the negation-as-failure operator is considered to be a *nonmonotonic* modal operator [Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1990; Lifschitz, 1991]. In particular, logic programs or deductive databases containing both negation as failure (*not*) and classical negation ($\neg$) can be interpreted as Reiter's default theories [Reiter, 1980] or disjunctive default theories [Gelfond *et al.*, 1991]. With these new semantics, logic

programming can be used as a powerful knowledge representation tool, whose applications contain reasoning with incomplete knowledge [Gelfond and Lifschitz, 1990; Gelfond and Lifschitz, 1991], expression of "don't-care" nondeterminism [Saccà and Zaniolo, 1990], exception handling [Kowalski and Sadri, 1990], default reasoning and abduction [Inoue, 1991a].

However, for these extended classes of logic programs, the top-down approach cannot be used for computation because there is no local property in evaluating programs. For example, there has been *no* top-down proof procedure which is sound with respect to the *stable model semantics* [Gelfond and Lifschitz, 1988] for general logic programs. Thus, we need *bottom-up* computation for correct evaluation of negation-as-failure formulas. This area is progressing, and there have been some proposals for computing stable models of general logic programs [Saccà and Zaniolo, 1990; Eshghi, 1990; Satoh and Iwayama, 1991]. Unfortunately, these previous approaches are only applicable to a simple (variable-free, disjunction-free) class of programs.

We show a bottom-up computation of *answer sets* of any class of function-free logic programs, including the *extended disjunctive databases* proposed by Gelfond and Lifschitz [1991] the proof procedure of which has not been found. Bottom-up computation reasons forwards, starting from unconditional literals, accumulating proved literals, and outputting models of programs. In evaluating $not\ P$ in a bottom-up manner, it is necessary to interpret $not\ P$ with respect to a fixpoint of computation because even if P is not currently proved, P might be proved in subsequent inferences. We thus come up with a completely different way of thinking for *not*: when we have to evaluate $not\ P$ in the current world, or a partial model, instead of computing "negation by failure to prove P", we split the world into two: (1) the world where P is assumed not to hold, and (2) the world where it is necessary that P holds. Each negation-as-failure formula $not\ P$ is thus translated into negative and positive literals with a modality expressing belief, i.e., "disbelieve P" and "believe P".

We thus provide a translation from any logic program (with negation as failure) into a *positive disjunctive program* (without negation as failure) [Minker, 1982] of which a *model generation theorem prover*, like SATCHMO [Manthey and Bry, 1988] or the MGTP [Fujita and Hasegawa, 1991], can compute the minimal models. Some pruning rules with respect to "believed" or "disbelieved" literals are expressed as integrity constraints that are dealt with by using object-level schemata on the MGTP [Koshimura and Hasegawa, 1991]. The MGTP then finds *all* answer sets *incrementally*, *without backtracking*, and *in parallel*. The proposed method is surprisingly simple and does not increase the computational complexity of the problem more than computation of the minimal models of positive disjunctive programs. The procedure has been implemented on top of the MGTP on a parallel inference machine, and has been applied to a legal reasoning system [Nitta *et al.*, 1992]. While we use the MGTP

to generate minimal models in this chapter, the proposed translation method may be linked with other methods to compute models [Fernández and Minker, 1991] or a fixpoint construction like [Reed *et al.*, 1991].

7.2 Positive Disjunctive Programs

This section shows how the MGTP [Fujita and Hasegawa, 1991] computes the minimal models of a *positive disjunctive program* [Minker, 1982], that is, a disjunctive database [Gelfond and Lifschitz, 1991] which contains neither negation as failure nor classical negation. The MGTP can deal with this class of programs. It is shown in later sections that every extended class of programs and databases can be translated to the class of positive disjunctive programs.

7.2.1 Minimal Models

A *positive disjunctive program* [Minker, 1982] is a set of rules of the form:

$$A_1 \mid \ldots \mid A_l \leftarrow A_{l+1}, \ldots, A_m\,, \tag{7.1}$$

where $m \geq l \geq 0$ and each A_i is an atom. According to [Gelfond and Lifschitz, 1991], we use the connective "|" instead of "$\vee$" although each A_i $(l \geq i \geq 1)$ is a disjunct of the consequent of the rule. When $l = 0$, a rule of the form:

$$\leftarrow A_1, \ldots, A_m$$

is called an *integrity constraint.* [1]

The meaning of a positive disjunctive program Σ can be given by the *minimal models* of Σ [Minker, 1982]. We represent the semantics in a similar way to the definition given by Gelfond and Lifschitz [Gelfond and Lifschitz, 1991], as follows. A rule containing variables stands for the set of its ground instances. We denote by $\mathcal{L}$ the set of ground literals in the language. An *answer set* of Σ is any minimal subset S of $\mathcal{L}$ satisfying the conditions:

1. For any ground instance $A_1 \mid \ldots \mid A_l \leftarrow A_{l+1}, \ldots, A_m$ $(l \geq 1)$ of any rule of Σ, if $A_{l+1}, \ldots, A_m \in S$, then for some i $(1 \leq i \leq l)$, $A_i \in S$;

2. For any ground instance $\leftarrow A_1, \ldots, A_m$ of any integrity constraint of Σ, if $A_1, \ldots, A_m \in S$, then $S = \mathcal{L}$.

[1] We allow for integrity constraints, that is, rules with empty consequents, in every class of logic programs and deductive databases. While this form of rules is not excluded by Gelfond and Lifschitz's definitions [1988; 1990; 1991], the corresponding semantics are not explicitly described.

We say Σ is *contradictory* if it has the answer set $\mathcal{L}$. It is easy to see that a contradictory program has the unique answer set $\mathcal{L}$. Unless a program Σ is contradictory, any answer set of Σ is a set of ground atoms, and the set of answer sets of Σ is equivalent to the set of minimal models of the program when each rule of the form (7.1) in Σ is identified with a clause (in the sense of classical logic) of the form:

$$A_1 \vee \ldots \vee A_l \vee \neg A_{l+1} \vee \ldots \vee \neg A_m \,. \tag{7.2}$$

Also, Σ is contradictory if and only if Σ is unsatisfiable in its clausal form.

7.2.2 MGTP

The answer sets or the minimal models of positive disjunctive programs can be computed by using the MGTP [Fujita and Hasegawa, 1991]. The MGTP is a parallel and refined version of SATCHMO [Manthey and Bry, 1988], which is a bottom-up model generation theorem prover that uses hyperresolution and case-splitting on non-unit derived clauses. In order to emphasize that the MGTP computes the models in a bottom-up manner, we express each rule of the form (7.1) in a positive disjunctive program as

$$A_{l+1}, \ldots, A_m \rightarrow A_1 \mid \ldots \mid A_l \,. \tag{7.3}$$

Given a positive disjunctive program Σ, the MGTP extends *model candidates* as follows. A model candidate is a subset of $\mathcal{L}$ and the initial set $\mathcal{S}_0$ of model candidates is given as $\{\emptyset\}$. Let $\mathcal{S}$ be a set of model candidates in any stage. The MGTP applies the following operation to $\mathcal{S}$ as long as possible:

> For any $S \in \mathcal{S}$ and any ground instance of any rule in Σ of the form (7.3),
> if $A_{l+1}, \ldots, A_m \in S$ and $A_1, \ldots, A_l \notin S$, then remove S from $\mathcal{S}$,
> and add $S \cup \{A_i\}$ to $\mathcal{S}$ for every $i = 1, \ldots, l$ $(l \geq 0)$.

If the MGTP cannot apply any operation to $\mathcal{S}$, it stops and returns $\mathcal{S}$.

In the above operation of the MGTP, we can deal with variables more elegantly. Instead of using ground instances of the rules, for each rule with variables in the form (7.3), we can obtain a substitution σ such that $A_{l+1}\sigma, \ldots, A_m\sigma$ $(l \geq 0)$ is satisfied in a model candidate S. We call the process of obtaining such a substitution σ a *conjunctive matching* of the antecedent literals against elements in S. Note that this process does not need full unification if the *range-restrictedness* condition is imposed on the rules [Manthey and Bry, 1988]. A rule is said to be *range-restricted* if every variable in the rule has at least one occurrence in its antecedents. Since every model candidate constructed by the MGTP in such a case contains only ground atoms, it is sufficient to consider one-way unification, i.e., matching, instead of full unification

with costly occurs check that does not allow circular bindings for variables. This is also a nice property for the implementation of the MGTP in KL1 (the kernel language for parallel inference machines developed at ICOT [Ueda and Chikayama, 1990]), because KL1 head unification is simply matching. The MGTP also improves the efficiency by removing redundant conjunctive matching with a *ramified-stack* algorithm [Fujita and Hasegawa, 1991].

When there are multiple rules whose antecedents are exactly the same, we want to avoid the same conjunctive matching more than once. Also, in the conversion techniques we present in later sections, we will use those rules whose consequents are disjunctions of conjunctions. For these purposes, the MGTP allows rules of the form:

$$A_{l+1}, \ldots, A_m \rightarrow A_{1,1}, \ldots, A_{1,k_1} \mid \ldots \mid A_{l,1}, \ldots, A_{l,k_l} , \tag{7.4}$$

where $m \geq l \geq 0$, $k_i \geq 1$ $(1 \leq i \leq l)$, and each A_i or $A_{i,j}$ is an atom. Each $A_{i,1}, \ldots, A_{i,k_i}$ represents a conjunction of atoms. We call a rule of this form (7.4) an *MGTP rule*. In summary, the MGTP operations are formally defined as follows. Let $\mathcal{S}$ be a set of model candidates in some stage, and S any element of $\mathcal{S}$.

1. **(Model candidate extension)** If there is an MGTP rule of the form

   $$A_{l+1}, \ldots, A_m \rightarrow A_{1,1}, \ldots, A_{1,k_1} \mid \ldots \mid A_{l,1}, \ldots, A_{l,k_l} \quad (l \geq 1)$$

   and a substitution σ such that $A_{l+1}\sigma, \ldots, A_m\sigma \in S$, and it does not hold that $A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma \in S$ for any $i = 1, \ldots, l$, then remove S from $\mathcal{S}$, and add $S \cup \{A_{i,1}\sigma, \ldots, A_{i,k_i}\sigma\}$ to $\mathcal{S}$ for all $i = 1, \ldots, l$;

2. **(Model candidate rejection)** If there is an MGTP rule of the form

   $$A_1, \ldots, A_m \rightarrow$$

   and a substitution σ such that $A_1\sigma, \ldots, A_m\sigma \in S$, then remove S from $\mathcal{S}$.

Given a set Σ of MGTP rules and an initial set $\mathcal{S}_0$ of model candidates (usually $\mathcal{S}_0 = \{\emptyset\}$), let $\mathcal{M}_\Sigma(\mathcal{S}_0)$ be the set of model candidates closed under the above two operations. Now, let us denote by $min(\mathcal{S})$ the set of minimal (in the sense of set inclusion of literals) model candidates of $\mathcal{S}$. Then, the output of the MGTP is characterized by the fixpoint operator $MGTP$: [2]

$$MGTP(\Sigma, \mathcal{S}_0) = min(\mathcal{M}_\Sigma(\mathcal{S}_0)) .$$

In the following, we assume that function symbols in the language are only constants and that the number of constants is finite. When a set Σ of MGTP rules

[2] A fixpoint semantics of the operator $\mathcal{M}_\Sigma$ is formally discussed by Inoue and Sakama [1992].

satisfies these assumptions as well as the range-restrictedness, we say Σ is *(finitely) groundable* [Bossu and Siegel, 1985]. It is easy to see that a finitely groundable program has a finite number of answer sets. For any finitely groundable set Σ of MGTP rules, the following properties hold. [3]

Proposition 7.1 If Σ is not contradictory, $MGTP(\Sigma, \{\emptyset\})$ is equivalent to the answer sets of Σ. □

Corollary 7.2 $MGTP(\Sigma, \{\emptyset\}) = \emptyset$ if and only if Σ is contradictory. □

It is guaranteed that a set of clauses has at least one minimal model if it is satisfiable [Bossu and Siegel, 1985]. The next is the basic property of the MGTP as a theorem prover.

Corollary 7.3 Suppose that every rule of the form (7.1) in Σ can be identified with the clause of the form (7.2). $MGTP(\Sigma, \{\emptyset\}) = \emptyset$ if and only if Σ is unsatisfiable. □

7.3 General Logic Programs

This section presents how to compute the answer sets of a *general logic program* [Gelfond and Lifschitz, 1988], that is, a logic program which contains negation-as-failure formulas but does not contain classical negation. While this class of logic programs is an instance of the more general class of disjunctive databases [Gelfond and Lifschitz, 1991] introduced in the next section, here we shall first explain the basic idea of bottom-up computation of negation as failure.

A *general logic program* is a set of rules of the form:

$$A_l \leftarrow A_{l+1}, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n \tag{7.5}$$

where, $n \geq m \geq l \geq 0$, $1 \geq l \geq 0$, and each A_i is an atom. The meaning of a general logic program is given by the *stable model semantics* [Gelfond and Lifschitz, 1988]. The semantics again treats a rule with variables as shorthand for the set of its ground instances. Let Π be a general logic program without variables. For any set of literals $S \subseteq \mathcal{L}$, let $\mathit{reduct}(\Pi, S)$ be the set of rules without *not* obtained from Π by removing

1. every rule containing a formula $\mathit{not}\, A$ in its body with $A \in S$, and
2. every negation-as-failure formula $\mathit{not}\, A$ in the bodies of the remaining rules.

[3] In this chapter, proofs of propositions and theorems are either deferred to Section 7.9 or omitted when they are obvious from the context or derivative from some other theorems.

Then, S is an *answer set* (or *stable model*) of Π if S is the answer set of $reduct(\Pi, S)$. Note that since $reduct(\Pi, S)$ is a program without containing negation as failure, its answer set is already defined by its minimal model in Section 7.2.1. Moreover, for any set S of literals, the answer set of $reduct(\Pi, S)$ is uniquely determined as there is no disjunction in any rule of $reduct(\Pi, S)$, while Π may have multiple answer sets.

Since we allow *integrity constraints*, i.e., rules without heads (expressed by $l = 0$ for the form (7.5)) in any general logic program Π, Π may be *contradictory*, that is, Π may have the answer set $\mathcal{L}$. Again, unless Π is contradictory, every stable model of Π is a set of ground atoms. Unlike positive disjunctive programs, a general logic program may not have any answer set. We say Π is *incoherent* if it has no answer set. Thus, any program is either a *consistent* program (which has consistent answer sets), contradictory program, or incoherent program [Inoue, 1991a].

Notice that the above definition of stable models is not constructive; S is defined by using itself in a way that a negation-as-failure formula $not\, P$ is true if P is not true in S. This S can be considered as a *guess* of a possible answer set. If S coincides with the smallest set of atoms deductively closed under the rules of Π, then the guess is correct so that it is acceptable as an answer set. Hence, the most direct way to compute the stable models of Π is to generate all possible guesses and then test if each guess is correct. This method is too explosive to realize because we have to generate and test $2^{|\mathcal{A}|}$ sets of atoms, where $\mathcal{A}$ is the set of ground atoms.

We thus want to make the number of guesses as few as possible. Let Π_P be the set of rules in Π which do not contain *not*, and Π_N the rest of the rules in Π. If Π has consistent stable models, then every stable model S should contain the least model M of Π_P. To compute the rest of the atoms in S, we make guesses as to whether each atom P appearing as $not\, P$ in Π_N is present. These guesses are delayed as long as possible: if there is a rule in which all antecedents that do not contain *not* are satisfied by a set S' of atoms such that $S' \supseteq M$, then we make a guess with respect to every $not\, P$ in the antecedents, and extend S' either by its consequent together with the guess that all such P's will not be present or by a guess that one of such a P will be derived.

Now, we are ready to compute the stable models of Π by using the MGTP. Based on the above discussion, we translate each rule in Π of the form (7.5):

$$A_l \leftarrow A_{l+1},\ \ldots,\ A_m,\ not\, A_{m+1},\ \ldots,\ not\, A_n$$

to the following MGTP rule:

$$A_{l+1},\ \ldots,\ A_m \rightarrow \neg \mathsf{K} A_{m+1}, \ldots, \neg \mathsf{K} A_n,\ A_l \mid \mathsf{K} A_{m+1} \mid \ \ldots\ \mid \mathsf{K} A_n\ . \tag{7.6}$$

The intuitive meaning of this MGTP rule is as follows. $\mathsf{K}A$ is a guess that A should hold, or A is *believed*, and $\neg\mathsf{K}A$ is a guess that A should not hold, or A is *disbelieved*.

In other words, $\mathsf{K}A$ ($\neg\mathsf{K}A$) imposes the condition that A must hold (A must not hold). For any MGTP rule of the form (7.6), if a model candidate S' satisfies $A_{l+1}, \ldots, A_m$, then S' is split into $n - m + l$ ($n \geq m \geq 0,\ 0 \leq l \leq 1$) model candidates.

We might relate the symbol K introduced in literals $\mathsf{K}A$, $\neg\mathsf{K}A$ with the modal operator K in an epistemic logic. We call these literals K-*literals*, and other literals without the symbol K *objective literals*. However, we avoid defining the symbol K either as a new connective or as a modal operator since we would like to remain within the MGTP calculus, or positive disjunctive programs, so that both positive and negative K-literals can be dealt with as atoms. To do so, we need to give the conditions which K-literals should satisfy. The following two schemata are thus introduced to reject model candidates when their guesses turn out to be wrong:

- If some ground instance of some atom A holds in S' and is not believed in S', then reject S'.

$$\neg\mathsf{K}A,\ A \rightarrow \qquad \text{for every atom } A\,. \tag{7.7}$$

- If some ground instance of some atom A is believed in S' and is not believed in S', then reject S'.

$$\neg\mathsf{K}A,\ \mathsf{K}A \rightarrow \qquad \text{for every atom } A\,. \tag{7.8}$$

Given a general logic program Π, we denote by $tr_1(\Pi)$ the set of rules consisting of the two schemata (7.7) and (7.8), and the MGTP rules obtained by replacing each rule (7.5) of Π with a rule (7.6).

The MGTP then computes the fixpoint $MGTP(tr_1(\Pi), \{\emptyset\})$ of model candidates. Each model candidate output by the MGTP contains K-literals as well as objective literals. Next is the condition that all of guesses made so far in a model candidate are correct. Let be $S' \in MGTP(tr_1(\Pi), \{\emptyset\})$.

- If any ground instance A of any atom is believed in S', then it must be true in S'.

$$\text{For every ground atom } A,\ \text{if } \mathsf{K}A \in S',\ \text{then } A \in S'\,. \tag{7.9}$$

We call condition (7.9) the $\mathbf{T}$-*condition*. It is named after axiom $\mathbf{T}$ in modal logic. That is, the model candidate in the fixpoint corresponds to a stable model only if each K-literal in it satisfies the condition. Note that the $\mathbf{T}$-condition is used as a test and cannot be a schema because our K-literals are merely guesses and we should confirm that A is actually derived from the program. Therefore, we *cannot* write the condition as:

$$\mathsf{K}A \rightarrow A \qquad \text{for every atom } A\,.$$

Now, for each model candidate S' computed by the MGTP, we denote the set of literals obtained from S' by removing all K-literals by $objective(S')$. The following two theorems guarantee that the above computation by the MGTP is sound and complete with respect to the stable model semantics.

Theorem 7.4 (1) If a model candidate S' in $MGTP(tr_1(\Pi), \{\emptyset\}) \neq \emptyset$ satisfies the **T**-condition (7.9), then $S = objective(S')$ is a stable model of Π. □
(2) If Π is consistent and S is a stable model of Π, then there is a model candidate S' in $MGTP(tr_1(\Pi), \{\emptyset\})$ such that $S = objective(S')$ and S' satisfies the **T**-condition (7.9). □

Theorem 7.5 (1) $MGTP(tr_1(\Pi), \{\emptyset\}) = \emptyset$ if and only if Π is contradictory. □
(2) Π is incoherent if and only if $MGTP(tr_1(\Pi), \{\emptyset\}) \neq \emptyset$ and there is no model candidate which satisfies the **T**-condition (7.9). □

In order that each MGTP rule of the form (7.6) may be *range-restricted*, in the original rule of the form (7.5) from which the MGTP rule is translated, every variable has at least one occurrence in its antecedents that do not contain *not*. This restriction is as natural as the range-restrictedness in positive disjunctive programs, and can be satisfied in most AI applications. At least, rules can be easily converted in order to satisfy this kind of range-restrictedness. The MGTP gives high inference rates for range-restricted rules by avoiding computation relative to their useless ground instances.

Example 7.6 Let the general logic program Π_1 consist of the following two rules:

$$P \leftarrow not\, Q\,,$$
$$Q \leftarrow not\, R\,.$$

These two rules are translated to the following MGTP rules:

$$\rightarrow \neg\mathsf{K}Q, P \mid \mathsf{K}Q\,,$$
$$\rightarrow \neg\mathsf{K}R, Q \mid \mathsf{K}R\,.$$

Then, $tr_1(\Pi_1)$ consists of these two rules and the two schemata (7.7) and (7.8). Now, let us see how the MGTP computes the stable models of Π_1. We start from the initial model candidates $\mathcal{S}_0 = \{\emptyset\}$.

1. $\mathcal{S}_1 = \{\, \{\neg\mathsf{K}Q, P\},\ \{\mathsf{K}Q\}\,\}$ by extending $\mathcal{S}_0$ with the first MGTP rule.

2. $\mathcal{S}_2 = \{\, S_1,\ S_2,\ S_3,\ S_4\,\}$, where $S_1 = \{\neg\mathsf{K}Q, P, \neg\mathsf{K}R, Q\}$,
$S_2 = \{\neg\mathsf{K}Q, P, \mathsf{K}R\}$, $S_3 = \{\mathsf{K}Q, \neg\mathsf{K}R, Q\}$, and $S_4 = \{\mathsf{K}Q, \mathsf{K}R\}$,
by extending $\mathcal{S}_1$ with the second MGTP rule.

3. $\mathcal{S}_3 = \{ S_2, S_3, S_4 \}$ by rejecting S_1 with the schema (7.7).
4. No operation is applicable to $\mathcal{S}_3$. Hence, $MGTP(tr_1(\Pi_1), \mathcal{S}_0) = \mathcal{S}_3$.
5. In $\mathcal{S}_3$, only S_3 satisfies the **T**-condition (7.9). Hence, $objective(S_3) = \{Q\}$ is the unique stable model of Π_1 by Theorem 7.4.

Note that when MGTP operations can be applied to model candidates with multiple rules, these rules can be processed *in any order*. Furthermore, computation is *incremental*. Consider, for example, that only the first rule of Π_1 is given at first. In this case, $\mathcal{S}_1$ is the output of the MGTP, in which only $\{\neg \mathsf{K}Q, P\}$ satisfies the **T**-condition (7.9), showing that $\{P\}$ is the stable model of the current program. Then, suppose that the second rule of Π_1 is added to the program that contains only the first rule. This time we can see that the MGTP outputs $\mathcal{S}_3$ (so that the stable model is again $\{Q\}$) by using $\mathcal{S}_1$ as the initial model candidates.

Example 7.7 Let the general logic program Π_2 consist of the following four rules:

$$R \leftarrow not\, R\,,$$
$$R \leftarrow Q\,,$$
$$P \leftarrow not\, Q\,,$$
$$Q \leftarrow not\, P\,.$$

These rules are translated to the following MGTP rules:

$$\rightarrow \neg \mathsf{K}R, R \,|\, \mathsf{K}R\,,$$
$$Q \rightarrow R\,,$$
$$\rightarrow \neg \mathsf{K}Q, P \,|\, \mathsf{K}Q\,,$$
$$\rightarrow \neg \mathsf{K}P, Q \,|\, \mathsf{K}P\,.$$

In this example, the first MGTP rule can be further reduced to

$$\rightarrow \mathsf{K}R\,,$$

if we prune the first disjunct by the schema (7.7). Therefore, the rule has computationally the same effect as the integrity constraint:

$$\leftarrow not\, R\,.$$

This integrity constraint says that every answer set has to contain R: namely, R *should be derived*. Now, it is easy to see that $MGTP(tr_1(\Pi_2), \{\emptyset\}) = \{ S_5, S_6, S_7 \}$, where $S_5 = \{\mathsf{K}R, \neg \mathsf{K}Q, P, \mathsf{K}P\}$, $S_6 = \{\mathsf{K}R, \mathsf{K}Q, \neg \mathsf{K}P, Q, R\}$, and $S_7 = \{\mathsf{K}R, \mathsf{K}Q, \mathsf{K}P\}$. The only model candidate that satisfies the **T**-condition (7.9) is S_6, showing that $\{Q, R\}$ is the unique stable model of Π_2. Note that the top-down procedure proposed by Eshghi and Kowalski is not sound [Eshghi and Kowalski, 1989, pp.251] because it has a top-down proof of P. In our case, we can easily check that $objective(S_5) = \{P\}$ is not a stable model because S_5 contains $\mathsf{K}R$ but does not contain R.

7.4 Extended Disjunctive Databases

An *extended disjunctive database* [Gelfond and Lifschitz, 1991] is a set of rules of the form:

$$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n \tag{7.10}$$

where $n \geq m \geq l \geq 0$ and each L_i is a literal. In particular, when an extended disjunctive database is a set of rules each of whose consequent consists of at most one literal:

$$L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n$$

($n \geq m \geq l \geq 0,\ 1 \geq l \geq 0$), it is called an *extended logic program* [Gelfond and Lifschitz, 1990].

Both extended logic programs and extended disjunctive databases allow for *classical negation* as well as negation as failure. Answer sets of extended disjunctive databases are defined as generalizations of both minimal models of positive disjunctive programs and stable models of general logic programs, as follows.

Let Σ be any extended disjunctive database without *not*. An *answer set* of Σ is any minimal subset S of $\mathcal{L}$ satisfying the conditions:

1. For any ground instance $L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m$ ($l \geq 1$) of any rule of Σ, if $L_{l+1}, \ldots, L_m \in S$, then for some i ($1 \leq i \leq l$), $L_i \in S$;

2. For any ground instance $\leftarrow L_1, \ldots, L_m$ of any integrity constraint of Σ, if $L_1, \ldots, L_m \in S$, then $S = \mathcal{L}$;

3. If for some ground atom A, $A \in S$ and $\neg A \in S$, then $S = \mathcal{L}$.

Now, let Π be any extended disjunctive database without variable. For any set $S \subseteq \mathcal{L}$, let $\mathit{reduct}(\Pi, S)$ be the set of rules without *not* obtained from Π by removing

1. every rule containing a formula $\mathit{not}\, L$ in its body with $L \in S$, and

2. every negation-as-failure formula $\mathit{not}\, L$ in the bodies of the remaining rules.

Then, S is an *answer set* of Π if S is one of the answer sets of $\mathit{reduct}(\Pi, S)$.

In the same way as general or extended logic programs [Inoue, 1991a], an extended disjunctive database is classified as either a *consistent*, *contradictory*, or *incoherent* database.

The effect of introducing classical negation into programs appears in the third condition of the above definition. Intuitively speaking, each answer set is a possible set of beliefs: each literal in an answer set can be considered to be true in the belief set. Since positive and negative literals have the same status, the result of negation by

failure to prove an atom A does not mean that its negation $\neg A$ is true. If an answer set contains neither A nor $\neg A$, the truth value of A is *unknown* in the belief set. Thus, the answer set semantics can provide for indefinite answers in answering queries, and such unknown information can be referred to in an extended disjunctive database. By using two kinds of negation, we can easily represent incomplete knowledge [Gelfond and Lifschitz, 1990], exceptions [Kowalski and Sadri, 1990], closed world assumptions [Gelfond and Lifschitz, 1990; Inoue, 1991a], defaults and hypotheses [Inoue, 1991a].

To compute the answer sets of an extended disjunctive database Π, we translate each rule in Π of the form (7.10):

$$L_1 \mid \ldots \mid L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n$$

to the following MGTP rule:

$L_{l+1}, \ldots, L_m \rightarrow$

$$\neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_1 \mid \ldots \mid \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_l \mid \mathsf{K} L_{m+1} \mid \ldots \mid \mathsf{K} L_n \,. \quad (7.11)$$

The next five schemata are introduced to reject model candidates containing wrong guesses. In below, when L is a literal, $\overline{L}$ is the literal complementary to L: for instance, when A is an atom, $\overline{A} = \neg A$ and $\overline{\neg A} = A$.

$$\neg \mathsf{K} L, L \rightarrow \quad \text{for every literal } L \in \mathcal{L} \,. \quad (7.12)$$

$$\neg \mathsf{K} L, \mathsf{K} L \rightarrow \quad \text{for every literal } L \in \mathcal{L} \,. \quad (7.13)$$

$$L, \overline{L} \rightarrow \quad \text{for every literal } L \in \mathcal{L} \,. \quad (7.14)$$

$$\mathsf{K} L, \overline{L} \rightarrow \quad \text{for every literal } L \in \mathcal{L} \,. \quad (7.15)$$

$$\mathsf{K} L, \mathsf{K} \overline{L} \rightarrow \quad \text{for every literal } L \in \mathcal{L} \,. \quad (7.16)$$

In the above schemata, (7.12) and (7.13) correspond to the two schemata (7.7) and (7.8) for general logic programs. The schema (7.14) is introduced to reflect the third condition of the definition of answer sets. Namely, if for some literal L and some substitution σ, a model candidate S' contains both $L\sigma$ and $\overline{L\sigma}$, then S' is rejected. The last two schemata, (7.15) and (7.16), are used for similar purposes. Now, we denote by $tr_2(\Pi)$ the set of rules consisting of the five schemata, (7.12), (7.13), (7.14), (7.15) and (7.16), and the MGTP rules obtained by replacing each rule (7.10) of Π with a rule (7.11).

Finally, the **T**-condition is as follows. Let be $S' \in MGTP(tr_2(\Pi), \{\emptyset\})$.

$$\text{For every ground literal } L \in \mathcal{L}, \text{ if } \mathsf{K} L \in S', \text{ then } L \in S' \,. \quad (7.17)$$

Theorem 7.8 Suppose that Π is consistent. Then,

$$min(\{\ objective(S') \mid S' \in MGTP(tr_2(\Pi), \{\emptyset\}),\\ S' \text{ satisfies the } \mathbf{T}\text{-condition (7.17) }\})$$

is equivalent to the answer sets of Π. □

When the program Π is not consistent, unlike general logic programs, we cannot identify whether Π is contradictory or incoherent. Here, we have weaker results.

Proposition 7.9 (1) If Π is contradictory, then $MGTP(tr_2(\Pi), \{\emptyset\}) = \emptyset$. □
(2) If $MGTP(tr_2(\Pi), \{\emptyset\}) \neq \emptyset$ and there is no model candidate satisfying the **T**-condition (7.17), then Π is incoherent. □

From the above result, when $MGTP(tr_2(\Pi), \{\emptyset\}) = \emptyset$, we can see that Π is either contradictory or incoherent. [4] Since Π is inconsistent anyway, this distinction is not very important if our goal is to compute consistent answer sets. Nevertheless, we could use the following property, which was first given by [Inoue, 1991a] (see Proposition 6.7 in Chapter 6).

Proposition 7.10 Let Π_P be the set of rules in Π not containing *not*. Π is contradictory if and only if Π_P is contradictory. □

Since Π_P is like a positive disjunctive program, the MGTP can easily compute the answer sets of Π_P by using the schema (7.14) so that we can determine whether the whole database Π is contradictory or not.

Example 7.11 Let us verify Theorem 7.8 in the example of the following extended disjunctive database Π_3, which is a slightly modified version of "broken-hand" example of Gelfond et al. [Gelfond *et al.*, 1991]:

$$Lh\text{-}Usable(x) \leftarrow Person(x),\ not\ Ab1(x)\ ,\\ Rh\text{-}Usable(x) \leftarrow Person(x),\ not\ Ab2(x)\ ,\\ Ab1(x) \leftarrow \neg Lh\text{-}Usable(x)\ ,\\ Ab2(x) \leftarrow \neg Rh\text{-}Usable(x)\ ,\\ Person(A) \leftarrow\ ,\\ \neg Lh\text{-}Usable(A) \mid \neg Rh\text{-}Usable(A) \leftarrow\ .$$

[4] The reason why the MGTP cannot output any model candidate with respect to $tr_2(\Pi)$ for an incoherent database Π is that the fourth and fifth schemata (7.15), (7.16) force the MGTP to prune model candidates that may either result in an inconsistent answer set $\mathcal{L}$ or violate the **T**-condition (7.17). Therefore, if these two schemata could be removed from the translated program, some model candidates would remain in the final output for the incoherent database. However, such a translation would reduce the efficiency of computation. See also the proof of Theorem 7.8 in Section 7.9.

Of these, the first two rules are translated to the following MGTP rules:

$$Person(x) \to \neg \mathsf{K} Ab1(x), Lh\text{-}Usable(x) \mid \mathsf{K} Ab1(x)\,,$$
$$Person(x) \to \neg \mathsf{K} Ab2(x), Rh\text{-}Usable(x) \mid \mathsf{K} Ab2(x)\,.$$

It is easy to see that $MGTP(tr_2(\Pi_3), \{\emptyset\})$ contains the following two model candidates that satisfy the **T**-condition (7.17):

$$\{\, Person(A),\ \neg Lh\text{-}Usable(A),\ Ab1(A),\ \mathsf{K} Ab1(A),\ \neg \mathsf{K} Ab2(A),\ Rh\text{-}Usable(A)\,\}\,,$$
$$\{\, Person(A),\ \neg Rh\text{-}Usable(A),\ Ab2(A),\ \mathsf{K} Ab2(A),\ \neg \mathsf{K} Ab1(A),\ Lh\text{-}Usable(A)\,\}\,.$$

Removing all the K-literals from these model candidates, we can get the two desired answer sets of Π_3.

7.5 Implementation of Schemata

7.5.1 Schemata on the MGTP

So far, we have represented schemata to reject model candidates that contain wrong guesses. To implement these schemata, we can simply use object-level schemata on top of the MGTP in the same way as an implementation of tableaux provers of modal logics by [Koshimura and Hasegawa, 1991]. For an atom A, the negative literal $\neg A$ can be expresses as `-`A, and for a literal L, the K-literal $\mathsf{K}L$ can be expressed as `k(`L`)` in KL1, where neither "`-`" nor "`k`" appears elsewhere in the program as a predicate symbol. The following is an example of expression of the five schemata for extended disjunctive databases.

```
(7.12)   -k(L), L --> false.
(7.14)   -A, A --> false.
(7.15)   k(A), -A --> false.
         k(-A), A --> false.
(7.16)   k(-A), k(A) --> false.
```

Note that the schema (7.13) for extended disjunctive databases can be omitted because it is an instance of the rule (7.14) above. Also, the two schemata (7.7) and (7.8) for general logic programs can be expressed by the above rules (7.12) and (7.14).

By using object-level schemata on the MGTP, we can use the MGTP without any change. This has a great advantage that the inference rules (logic) and the inference engine (control) can be clearly separated. However, to improve the efficiency, we can consider another method as shown next.

7.5.2 Restriction of Model Candidate Extensions

In Section 7.2.2, we have defined the model candidate extension operation for an MGTP rule of the form (7.4). This operation does not distinguish K-literals from objective literals. However, to improve the efficiency, we can incorporate the schemata into the operation so that extensions should be avoided if the resultant model candidates are to be pruned immediately. For example, consider the MGTP rule (7.11) obtained by translating from a rule (7.10) in an extended disjunctive database:

$$L_{l+1}, \ldots, L_m \rightarrow \\ \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_1 \mid \ldots \mid \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n, L_l \mid \mathsf{K} L_{m+1} \mid \ldots \mid \mathsf{K} L_n \,,$$

where $n \geq m \geq l \geq 0$. For this rule and a model candidate $S' \in \mathcal{S}$, the model candidate extension by the MGTP works as follows:

> **If** for some substitution σ, $L_{l+1}\sigma, \ldots, L_m\sigma \in S'$,
> it does not hold that $\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma \in S'$ for any $i = 1, \ldots, l$,
> and $\mathsf{K} L_j\sigma \notin S'$ for any $j = m+1, \ldots, n$,
> **then** remove S' from $\mathcal{S}$,
> add $S' \cup \{\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma\}$ to $\mathcal{S}$ for all $i = 1, \ldots, l$,
> and add $S' \cup \{\mathsf{K} L_j\sigma\}$ to $\mathcal{S}$ for all $j = m+1, \ldots, n$.

On the other hand, a "forward-checking" version is as follows:

> **If** for some substitution σ, $L_{l+1}\sigma, \ldots, L_m\sigma \in S'$,
> $L_i\sigma \notin S'$ for any $i = 1, \ldots, l$,
> and $\mathsf{K} L_j\sigma, L_j\sigma \notin S'$ for any $j = m+1, \ldots, n$,
> **then** remove S' from $\mathcal{S}$,
> add $S' \cup \{\neg \mathsf{K} L_{m+1}\sigma, \ldots, \neg \mathsf{K} L_n\sigma, L_i\sigma\}$ to $\mathcal{S}$ for each i $(1 \leq i \leq l)$ such that
> $\neg \mathsf{K} L_i\sigma, \overline{L_i\sigma}, \mathsf{K}\overline{L_i\sigma} \notin S'$ and that $L_i\sigma \neq L_j\sigma$ for any $j = m+1, \ldots, n$,
> and add $S' \cup \{\mathsf{K} L_j\sigma\}$ to $\mathcal{S}$ for each j $(m+1 \leq j \leq n)$ such that
> $\neg \mathsf{K} L_j\sigma, \overline{L_j\sigma}, \mathsf{K}\overline{L_j\sigma} \notin S'$.

The five schemata for extended disjunctive databases are completely incorporated into the above new model candidate extension operation, which checks whether each new objective or K-literals can be safely added or not to S'.[5] At the expense of these extra checking, we can reduce the number of model candidate extensions. In practice, the cost of generating or extending model candidates and keeping them is much more expensive than the cost of these extra checking. Moreover, we can dispense with conjunctive matching of antecedents of the schemata, which is always tried against

[5] A forward-checking mechanism will work automatically by using a new version of the MGTP (the "lazy" MGTP), which does not extend model candidates to be pruned by some integrity constraints.

any model candidate even if it is not to be pruned. Notice also that in the above new operation, a model candidate S' will never be extended if it contains a literal $L_i\sigma$ for any $i = 1, \ldots, l$ or any $i = m+1, \ldots, n$. This is because, if S' contains an $L_i\sigma$ $(1 \leq i \leq l)$, then it can not be extended in such a way that new objective literals are added as fewer as possible, and if S' contains an $L_j\sigma$ $(m+1 \leq j \leq n)$, then its extension will either be pruned or never increase a new objective literal.

7.6 Discussion

In this section, we compare the proposed method to other approaches that evaluate logic programs containing negation-as-failure formulas. The proposed method has several computational advantages: in a word, it can find *all* answer sets for *every* class of groundable logic programs or disjunctive databases, *incrementally*, *without backtracking*, and *in parallel*. We shall examine these characteristics as follows.

1. *Finding all answer sets.*
   This fact means that the proposed method is *complete* with respect to the answer set semantics. This is due to the fact that the MGTP [Fujita and Hasegawa, 1991] can find every minimal model of a positive disjunctive program. For positive disjunctive programs, bottom-up computation has recently been recognized to be more useful than top-down computation, and there has been some other methods to compute minimal models [Fernández and Minker, 1991][6] or to characterize fixpoint computation [Reed *et al.*, 1991]. Therefore, our translation method may be linked with those methods as well as the bottom-up SATCHMO [Manthey and Bry, 1988] prover. Moreover, our method is *sound* with respect to the answer set semantics, again due to bottom-up computation. Top-down computation, on the other hand, can never guarantee the soundness even for general logic programs as shown by [Eshghi and Kowalski, 1989].

2. *Applicable to every class of logic programs or deductive databases.*
   Several procedures have been proposed to compute the stable models of general logic programs [Saccà and Zaniolo, 1990; Eshghi, 1990; Satoh and Iwayama, 1991]. However, these procedures deal only with the propositional case and none of them can be extended to allow disjunctive databases because they are based on TMS-like algorithms. Our procedure, on the other hand, can deal with both variables and disjunctions for range-restricted rules very elegantly. Note also

[6] Fernández and Minker [1992] also extend their model generation method to general disjunctive databases (without classical negation) by using a similar but different transformation method. See [Inoue and Sakama, 1992] for the difference between their approach and ours.

that the proposed method does not increase the computational complexity of the problem more than computation of the minimal models of positive disjunctive programs; the size of the translated MGTP rules is the same as the size of the original rules, and the disjuncts introduced by the translation would be of the same size of the positive literals in the heads of the positive disjunctive database if each negation-as-failure formula *not* A were moved to the head as A, that is, *not* A were replaced with the literal $\neg A$ that is complementary to A in the sense of classical clausal logic.

3. *Incremental, backtrack-free computation.*
   Since we keep K-literals in each model candidate, when new rules are added to the database, the previous set of model candidates can be used as the input to the next computation. Procedures given by [Saccà and Zaniolo, 1990; Satoh and Iwayama, 1991] cannot be used incrementally. Eshghi's [1990] proposal may be used incrementally, but it requires an exponential-time algorithm to convert its data structures into the stable models, which is much more complicated than our use of the **T**-condition (7.9).
   Furthermore, by means of case-splitting of the MGTP, a model candidate is split into multiple model candidates. While this is the place where nondeterminism arises in our procedure, those model candidates can be dealt with independently without future backtracking. This means that, for every generated model candidate, each ground instance of any rule is evaluated only once. Note that we do not assume any control strategy for the processing of the model candidates: we can use either depth-first, breadth-first, or best-first search strategies. Even if we use sequential processing with a depth-first manner, backtracking to another model candidate is not very costly. By this control-independence property, we need not enumerate rules for their applications to model candidate extensions. On the other hand, nondeterministic procedures of [Saccà and Zaniolo, 1990; Satoh and Iwayama, 1991] rely on selections of rules to be applied first so that backtracking involves reevaluation of rules and its cost is very high.

4. *Parallel implementation.*
   Our method is also a first attempt to compute answer sets in parallel. The procedure has been implemented on a distributed-memory multiprocessor machine, Multi-PSI, developed at ICOT. The translation is especially suitable for OR-parallelism because, for each negation-as-failure formula, we will make guesses to believe or disbelieve it. Multiple model candidates are thus taken as the source for exploiting OR-parallelism of the MGTP.

7.7 Application to Legal Reasoning

The proposed method has been applied to a legal reasoning system [Nitta *et al.*, 1992]. We can see some advantages of the proposed method from the viewpoint of this application.

It has been recognized that to use two kinds of negation, negation as failure and classical negation, are very powerful in order to represent knowledge of legislation in logic programs [Kowalski and Sadri, 1990; Sartor, 1991]. By [Sartor, 1991], a *primary* fact (whose proof has to be demonstrated) can be represented by a literal, while a *secondary* fact L (for which proof to the contrary must not be given) as a negation-as-failure formula $not\,\overline{L}$. The question is how we should use logic programming in legal reasoning. Usually, in a legal reasoning system, a set of ground facts and a set or general rules or norms are given, and the goal is to obtain the possible *interpretations* containing judicial precedents. However, such an inference is plausible as it is often necessary to reason with incomplete information. Therefore, the system should create explanations or justifications why the conclusions have been derived under some legal concept. This kind of processes sometimes reflects antagonistic arguments by a jury in a court.

For the above purposes, our computation is extremely desirable. Bottom-up computation constructs the model candidates, each of which corresponds to a possible interpretation. In each model candidate, a negative K-literal $\neg\mathsf{K}L$ represents an absence of the contrary to a secondary fact $\overline{L}$. Thus, if a proof for L could be given against $\neg\mathsf{K}L$, then the corresponding argument would be rebuted. On the other hand, since the objective literals in a model candidate that does not satisfy the **T**-condition is not an answer set, the model candidate can be understood as a weak argument. In this case, though, we can see that if only a literal L could be established for each positive K-literal $\mathsf{K}L$ that has not been justified in the model candidate, such an argument might become valid. Hence, those extra information represented by K-literals in model candidates can play an important role in legal reasoning.

7.8 Summary

In this chapter, we have presented a novel technique to compute answer sets of logic programs or disjunctive databases. The technique is simply based on a bottom-up model generation method for positive disjunctive databases, together with integrity constraints over K-literals expressed by object-level schemata on the MGTP. The proposed translation is also very simple and does not increase the program size. Moreover, the method has been implemented on a parallel inference machine.

We should comment, though, that while our results have a useful application to legal reasoning, the general question of how to avoid combinatorial explosion in constructing model candidates still needs to be investigated. From our experience in testing our procedure for some kinds of nonmonotonic reasoning and planning, it is observed for some applications that a query-answering mechanism is necessary in addition to computation of the model candidates. We also expect that for semantics of logic programs or disjunctive databases which are different from the answer set semantics, it may be possible to apply different translations into MGTP rules containing K-literals together with different integrity constraints. These issues will be discussed in a separate paper.

7.9 Proofs

Here, we give a proof of Theorem 7.8. Since Theorem 7.4 is an instance of Theorem 7.8, we omit its proof. We will also omit proofs of other theorems and propositions as they can be proved more easily. In the following, Π is a finitely groundable extended disjunctive database. We can further assume without loss of generality that Π is a set of ground rule of the form (7.10). We use the following notations throughout this section. For each rule R in Π of the form (7.10):

$$R \;=\; L_1 \,|\, \ldots \,|\, L_l \leftarrow L_{l+1},\, \ldots,\, L_m,\, not\, L_{m+1},\, \ldots,\, not\, L_n\,,$$

we define the following:

$$\begin{aligned}
tr(R) &= L_{l+1},\, \ldots,\, L_m \rightarrow \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n,\, L_1 \,|\, \ldots \\
&\qquad \ldots \,|\, \neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n,\, L_l \,|\, \mathsf{K} L_{m+1} \,|\, \ldots \,|\, \mathsf{K} L_n\,, \\
red(R) &= L_{l+1},\, \ldots,\, L_m \rightarrow L_1 \,|\, \ldots \,|\, L_l\,, \\
dis(R) &= \{L_1, \ldots, L_l\}\,, \\
pos(R) &= \{L_{l+1},\, \ldots,\, L_m\}\,, \\
neg(R) &= \{L_{m+1}, \ldots, L_n\}\,, \\
\mathsf{K} neg(R) &= \{\mathsf{K} L_{m+1}, \ldots, \mathsf{K} L_n\}\,, \\
\neg \mathsf{K} neg(R) &= \{\neg \mathsf{K} L_{m+1}, \ldots, \neg \mathsf{K} L_n\}\,.
\end{aligned}$$

Given Π and a set of literals $S \subseteq \mathcal{L}$, we define:

$$reduct*(\Pi, S) = \{\; red(R) \mid R \in \Pi,\, neg(R) \cap S = \emptyset \;\} \cup \{\; \text{the schema (7.14)} \;\}\,.$$

Lemma 7.12 Let be $S' \in MGTP(tr_2(\Pi), \{\emptyset\})$. If $S = objective(S')$ is a minimal set such that S' satisfies the **T**-condition (7.17), then $S \in MGTP(reduct*(\Pi, S), \{\emptyset\})$.

Proof: For every rule $tr(R)$ in $tr_2(\Pi)$, it holds that, if $pos(R) \subseteq S'$, then either $\mathsf{K}neg(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $\neg\mathsf{K}neg(R) \subseteq S'$. Since S' was not pruned by the schema (7.12) and satisfies the **T**-condition (7.17), it holds that for every rule $tr(R)$ in $tr_2(\Pi)$, if $pos(R) \subseteq S'$, then either $neg(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $neg(R) \cap S' = \emptyset$. Now, for each rule $tr(R)$ in $tr_2(\Pi)$, there is a rule R in Π. Therefore, for every rule R in Π, if $pos(R) \subseteq S$, then either $neg(R) \cap S \neq \emptyset$, or $dis(R) \cap S \neq \emptyset$ and $neg(R) \cap S = \emptyset$. Because of the minimality of S and the closedness of the MGTP operations, $S \in MGTP(reduct*(\Pi, S), \{\emptyset\})$. □

Note that by the above proof, only the two schema, (7.12) and (7.14), and the **T**-condition (7.17) are necessary to guarantee the soundness of computation; other schemata are used for obtaining the efficiency of computation.

Lemma 7.13 Let be $S \in MGTP(reduct*(\Pi, S), \{\emptyset\})$. If Π is consistent, then S is a minimal set satisfying the condition: S is the objective literals of some model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that S' satisfies the **T**-condition (7.17).

Proof: For every rule R in $reduct*(\Pi, S)$, it holds that, if $pos(R) \subseteq S$, then $dis(R) \cap S \neq \emptyset$. We then see that for every rule R in Π, if $pos(R) \subseteq S$, then either $neg(R) \cap S \neq \emptyset$, or $dis(R) \cap S \neq \emptyset$ and $neg(R) \cap S = \emptyset$. Now, for each rule R in Π, there is a rule $tr(R)$ in $tr_2(\Pi)$. Since $MGTP(reduct*(\Pi, S), \{\emptyset\})$ is closed under the MGTP operations, S is a minimal set satisfying the condition: there exists a model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that $S = objective(S')$ and that for any rule $tr(R)$ in $tr_2(\Pi)$, it holds that, if $pos(R) \subseteq S'$, then either $\mathsf{K}neg(R) \cap S' \neq \emptyset$, or $dis(R) \cap S' \neq \emptyset$ and $\neg\mathsf{K}neg(R) \subseteq S'$. It is easy to check that this S' is not pruned by the schemata, and that S' satisfies the **T**-condition (7.17). □

Theorem 7.8 Suppose that Π is consistent. The answer sets of Π is equivalent to:

$$min(\{\ objective(S') \mid S' \in MGTP(tr_2(\Pi), \{\emptyset\}),\\ S' \text{ satisfies the } \mathbf{T}\text{-condition (7.17) }\}).$$

Proof: A set $S \subseteq \mathcal{L}$ is an answer set of Π if S is an answer set of $reduct(\Pi, S)$.

$\Leftrightarrow S \in MGTP(reduct*(\Pi, S), \{\emptyset\})$ (by a variant of Proposition 7.1).

$\Leftrightarrow S$ is a minimal set satisfying the condition: S is the objective literals of some model candidate S' in $MGTP(tr_2(\Pi), \{\emptyset\})$ such that S' satisfies the **T**-condition (7.17) (by Lemmas 7.12 and 7.13). □

Chapter 8

Conclusion and Future Direction

In this dissertation, we have presented several logical frameworks for hypothetical reasoning, and have shown how these different frameworks are related.

In Chapter 2, we started with a survey of existing logical frameworks of abduction and nonmonotonic reasoning, and presented the relationships between these frameworks. Previous work has shown that Reiter's default logic is a very important vehicle for such investigation: abduction can be formalized by using prerequisite-free normal default theories as credulous prediction, and logic programs with negation as failure can be formalized by using generic default theories. We discussed the three remaining problems of hypothetical reasoning in Section 2.3.4, that is,

1. the relationship between abduction and deduction (for enabling efficient computation of abduction),
2. the relationship between abduction and circumscription (for enabling computation of skeptical prediction), and
3. the relationship between abduction and logic programs (for enabling computation of generic defaults).

The remaining chapters were thus devoted to detailed analysis of these problems:

1. In Chapter 3, we characterized abduction in terms of a consequence-finding problem in automated deduction. This characterization enables efficient theorem proving techniques to be utilized for computation of explanations. An effective procedure, SOL-resolution, which is complete for finding the new characteristic clauses, was proposed in Chapter 4.
2. In Chapter 5, we characterized theorem proving in circumscription as finding some combinations of abductive explanations, and developed an efficient mechanism for obtaining answers to queries.

3. In Chapter 6, we proposed a framework of hypothetical reasoning (knowledge system) within extended logic programs, for default reasoning and exceptions, inconsistency resolution, closed world assumption, and abduction. We have also shown that any knowledge system can be transformed into an extended logic program. This framework can be computed by using a model generation theorem prover, based on a fixpoint characterization of these extended classes of logic programs and disjunctive databases, as presented in Chapter 7.

In summary, we have completed and strengthened the ties between abduction and nonmonotonic reasoning. Both types of default prediction, credulous and skeptical, are now characterized by abduction, and can be computed by a consequence-finding procedure. A class of more generic defaults can be handled by hypothetical reasoning within logic programs, and can be computed using a model generation procedure. Therefore, we conclude that the links between hypothetical reasoning and logical frameworks for nonmonotonic reasoning are bidirectional: hypothetical reasoning can be formalized by using some nonmonotonic logics, and some nonmonotonic logics can be computed by using hypothetical reasoning.

In the rest of this chapter, apart from representational issues for abductive and nonmonotonic reasoning, we focus on computational mechanisms. Since we have characterized various kinds of hypothetical reasoning in terms of classical logic, all we need is efficient deductive mechanisms for computation. We thus look again at the two novel architectures, *consequence-finding* and *model generation*, each of which has been proposed in this dissertation and is a core technology in the computation of various hypothetical reasoning. We shall regard these two architectures not only as theorem proving procedures but as generic problem solving systems so that we have a better understanding of the computational properties of hypothetical reasoning. Finally, we will examine some interesting issues for future research by taking a look at the further possibilities of these architectures.

8.1 Consequence-Finding

We have already seen that many reasoning problems of interest in AI can be characterized by consequence-finding. These reasoning problems contain *abduction* (Sections 3.3.1 and 4.2.2), *CMS/ATMS* (Sections 3.3.3 and 4.2.4), *circumscription* (Sections 3.3.4 and 5.2.2), and their applications to such as diagnosis, synthesis (planning, design), natural language understanding, intelligent databases, and commonsense reasoning (Section 3.3.5). In Chapters 4 and 5, a number of key technologies have been proposed for computation of (new) characteristic clauses, such as subsumption, skip preference, compilation, and answer extraction.

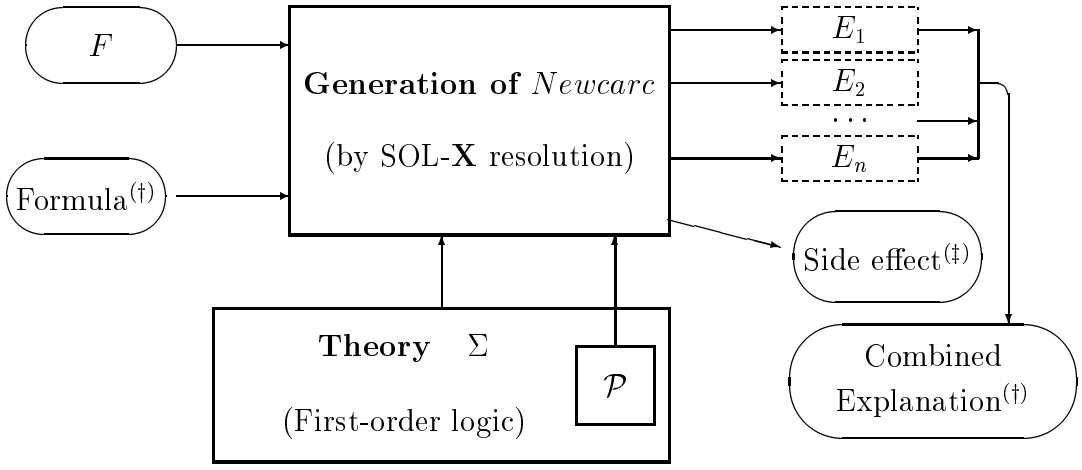

Figure 8.1: A consequence-finding architecture

Figure 8.1 illustrates a general problem solving architecture that is based on the consequence-finding procedures defined in Chapter 4. For this architecture, we are given a theory Σ which is a set of first-order formulas and a production field $\mathcal{P}$. The basic inference engine utilizes SOL-resolution (Section 4.2.1) or its variants (Section 4.3) to produce the (new) characteristic clauses. We may use this architecture in various ways:

1. Without any input formulas other than the axiom set Σ, the characteristic clauses $Carc(\Sigma, \mathcal{P})$ can be computed incrementally by adding each axiom in Σ one by one.

2. When a formula F is newly input to Σ, ramification can be computed as the new characteristic clauses $Newcarc(\Sigma, F, \mathcal{P})$.

3. In abduction, the new formula F is the negation of the observation. The system outputs negations of minimal explanations $E_1, \ldots, E_n$.

4. If necessary, such explanations are combined appropriately, then a weaker explanation (†) is formed.

5. In circumscription, the negation of such a combined explanation is then input to the architecture (†). We can check whether this new formula has an explanation (‡) or not. If it is not explainable, then the original query is a theorem of the circumscription; otherwise, we look for another combination.

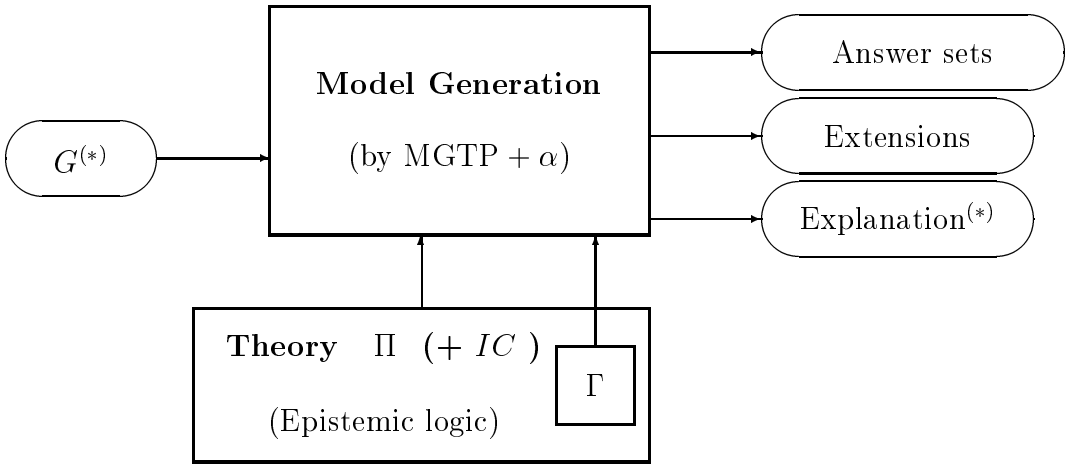

Figure 8.2: A model generation architecture

6. In plan recognition problems, each explanation of the observation is input to the architecture (†) without taking its negation, and we can see what side effect is caused by this explanation (‡). This computation can be done by taking another production field $\mathcal{P}'$. If any unintended side effect is caused, then we look for another explanation.

It may be possible to apply this architecture to other interesting problems which have not yet been recognized well, such as analogy, induction and other non-deductive inference mechanisms.

8.2 Model Generation

We have seen that many important problems for default and abductive reasoning can be formalized by theory formation within extended classes of logic programs and disjunctive databases. By applying the knowledge system framework, we can represent *nonmonotonic databases*, *defaults/exceptions*, *nonmonotonic (A)TMSs*, and *abduction* (Section 6.6). We have also provided a transformation from any knowledge system into an extended logic program (Chapter 6), and another transformation from any logic program with negation as failure into a disjunctive program with integrity constraints but without negation as failure (Chapter 7) so that a model generation theorem prover can compute its answer sets.

Figure 8.2 shows a general architecture for problem solving based on the model generation procedures defined in Chapter 7. For this architecture, we are given a

theory Π and integrity constraints IC each of which is a set of rules with some modality, and a set of hypotheses Γ. The basic inference engine is of course a model generation theorem prover such as the MGTP (Section 7.2.2) or its optimized version for some special purposes. We may use this architecture in the following ways:

1. When the theory Π contains formulas with special properties, such as nonmonotonic (un)provability (*negation as failure*) and *consistency* of abductive explanations, these formulas are translated into some MGTP rules with a kind of modality so that the MGTP can deal with them within classical logic (for abduction, the transformation method will be described later in Section 8.3.1). Extra requirements for these special properties are thus reduced to "generate-and-test" problems of model candidates which can be handled by the MGTP very efficiently through case-splitting of non-unit consequences and rejection of inconsistent model candidates.

2. Without any input formulas other than the axiom set Π, the minimal models of Π can be constructively computed.

3. When a query G is provided, we look for an answer set which satisfies the query. For logic programs and disjunctive databases, this computation can be done by supplying the integrity constraint

$$\leftarrow \mathit{not}\, G\,.$$

4. In abduction, we look for an answer set which satisfies G by allowing the formula from Γ to be hypothesized. Those formulas used to derive G form its explanation $(*)$.

It may be possible to apply this architecture to other non-standard logics such as temporal logic and other modal logics. The proof method of the MGTP is very close to the tableaux proof procedures, as each rule of tableaux can be represented by an MGTP rule in a direct manner (see [Koshimura and Hasegawa, 1991]). A close condition of tableaux methods is represented by a negative clause (i.e., an integrity constraint), and a decomposition rule is represented by a mixed clause.

8.3 Further Possibilities of Model Generation

We have explained two architectures for computing abductive and nonmonotonic reasoning. The *consequence-finding* architecture is used for those hypothetical reasoning frameworks that are based on classical first-order logic with meta information (production field). SOL-resolution and its variants are extensions of *top-down*,

backward-chaining theorem proving procedures (i.e., C-ordered linear resolution). Note that while SOL-resolution itself is a top-down procedure, generation of ramification ($Newcarc$) is in a form of forward reasoning in the sense that "ramifications are generated by reasoning forward from goals created in the course of goal reduction" [Finger, 1987].

On the other hand, the *model generation* architecture is used for those hypothetical reasoning frameworks that are based on non-standard logics. In these frameworks, integrity constraints relative to the epistemic status can be directly dealt with by these logics. The MGTP is a *bottom-up*, *forward-chaining* model construction procedure based on hyperresolution and case-splitting.

Both of these reasoning styles have merits and demerits, which are complementary to each other. In this section, we thus consider some important possible extensions of work presented in this dissertation by combining these two approaches.

8.3.1 Bottom-Up Abduction

As stated in Section 2.3.1, abduction is usually implemented by an extension of a top-down theorem-proving procedure. For example, Theorist [Poole *et al.*, 1987] and SOL-resolution [Inoue, 1991b] are extensions of the Model Elimination procedure [Loveland, 1978]. This is why we consider the consequence-finding architecture to be suitable for abduction.

However, there is nothing to prevent us from using a bottom-up procedure to implement abduction. In fact, we have developed the abductive reasoning system APRICOT/0 [Ohta and Inoue, 1990; Ohta and Inoue, 1991a; Ohta and Inoue, 1991b], which consists of a forward-chaining inference engine and the ATMS [de Kleer, 1986]. The ATMS is used to keep track of the results of inference in order to avoid both repeated proofs of subgoals and duplicate proofs among different hypotheses deriving the same subgoals.

Here, we introduce possible realizations of abductive reasoning systems built on the bottom-up theorem prover MGTP. In the following, we consider the first-order abductive framework (Σ, Γ), where Σ is a set of groundable Horn clauses and Γ is a set of atoms (*abducibles*). We have already developed several parallel abductive systems [Inoue *et al.*, 1992b] using the MGTP. We outline four of them below.

1. **MGTP+ATMS** (Figure 8.3).

   This is a parallel implementation of APRICOT/0 [Ohta and Inoue, 1990] which utilizes the ATMS to check consistency. The MGTP is used just as a forward-chaining inference engine, and the ATMS keeps a current set of beliefs $\mathbf{M}$, in which each ground atom is associated with some hypotheses.

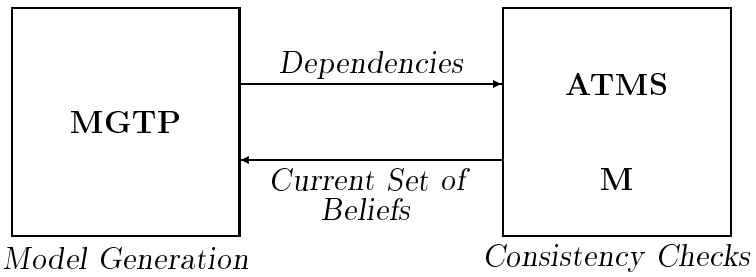

Figure 8.3: MGTP+ATMS

Parallelism is exploited by executing the parallel ATMS. However, because there is only one channel between the MGTP and the ATMS, the MGTP often has to wait for the results of the ATMS. Thus, the effect of parallel implementation is limited.

2. **MGTP+MGTP** (Figure 8.4).

   This is a parallel version of the method described in [Stickel, 1991]. In addition, consistency is checked by calling another MGTP ($MGTP2$). In this system, each hypothesis H in Γ is represented by $fact(H, \{H\})$, and each Horn clause in Σ of the form:

   $$A_1 \wedge \ldots \wedge A_n \supset C\,,$$

   is translated into the MGTP rule of the form:

   $$fact(A_1, E_1),\ \ldots,\ fact(A_n, E_n) \rightarrow fact(C,\ \mathsf{cc}(\bigcup_{i=1}^{n} E_i))\,,$$

   where E_i is a set of hypotheses from Γ on which A_i depends, and the function cc is defined as:

   $$\mathsf{cc}(E) = \begin{cases} E & \text{if } \Sigma \cup E \text{ is consistent;} \\ \mathsf{nil} & \text{otherwise.} \end{cases}$$

   A current set of beliefs $\mathbf{M}$ is kept in the form of $fact(A, E)$ representing a meta-statement that $\Sigma \cup E \models A$, but is stored in the inference engine ($MGTP1$) itself. Each time $MGTP1$ derives a new ground atom, the consistency of the combined hypotheses is checked by $MGTP2$.

   The parallelism comes from calling multiple $MGTP2$'s at once. This system achieves more speed-up than the MGTP+ATMS method. However, since $MGTP1$ is not parallelized, the effect of parallelization depends heavily on how much consistency checking can be performed in parallel at once.

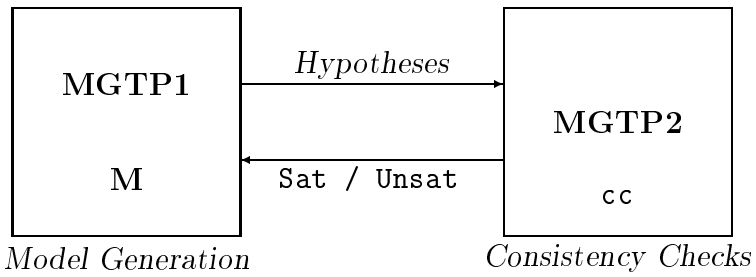

Figure 8.4: MGTP+MGTP

3. **All Model Generation Method**.

   No matter how good the MGTP+MGTP method might be, the system consists of two different components. The possibilities for parallelization therefore remain limited. In contrast, the following two "model generation" methods do not separate the inference engine from consistency checking, but realize both functions in only one MGTP. In such methods, the MGTP is used not only as an inference engine but also as a "generate-and-test" mechanism so that consistency checks are automatically performed. For this purpose, we can utilize the capability of extension and rejection of model candidates supplied by the MGTP. Therefore, multiple model candidates can be kept in distributed memories instead of keeping the one global belief set **M** used in the above two methods, thus a great amount of parallelism can be obtained.

   The all model generation method is the most direct way to implement reasoning with hypotheses. For each hypothesis H in Γ, we supply a rule of the form:

$$\rightarrow H \mid \neg \mathsf{K} H \,, \tag{8.1}$$

   where $\neg \mathsf{K} H$ means that "H is not assumed to be true in the model". Namely, each hypothesis is assumed either to hold or not to hold. Since this system may generate $2^{|\Gamma|}$ model candidates, the method is often explosive for a number of practical applications.

4. **Skip Method**.

   To reduce the number of generated model candidates as much as possible, we can use a method that delays the case-splitting for each hypothesis. This approach is similar to the processing of negation as failure with the MGTP [Inoue *et al.*, 1992a] introduced in Chapter 7. That is, we do not supply any rule of the form (8.1) for any hypothesis of Γ, but, instead, introduce hypotheses when

they are necessary. When a clause in Σ contains the negative occurrences of abducible predicates $H_1, \ldots, H_m$ ($H_i \in \Gamma$, $m \geq 0$) and is in the form:

$$A_1 \wedge \ldots \wedge A_l \wedge \underbrace{H_1 \wedge \ldots \wedge H_m}_{abducibles} \supset C\,,$$

we translate it into the following MGTP rule:

$$A_1, \ldots, A_l \rightarrow H_1, \ldots, H_m, C \mid \neg \mathsf{K} H_1 \mid \ldots \mid \neg \mathsf{K} H_m\,. \tag{8.2}$$

In this translation, each hypothesis in the premise part is *skipped* instead of being resolved, and is moved to the right-hand side. This operation is a counterpart of the **Skip** rule in the top-down approach defined for SOL-resolution in Chapter 4. Just as in the schema (7.7) for negation as failure, a model candidate containing both H and $\neg \mathsf{K} H$ is rejected by the schema:

$$\neg \mathsf{K} H,\ H \rightarrow \qquad \text{for every hypothesis } H\,.$$

Some evaluation of these abductive systems that are applied to planning and design problems are described in [Inoue *et al.*, 1992b]. Although we need to investigate further how to avoid possible combinatorial explosion in constructing model candidates for the skip method, we conjecture that (a variant of) the skip method will be the most promising from the viewpoint of parallelism.

Another, and perhaps the most important, advantage of the skip method is that it may easily be combined with negation as failure so that knowledge bases can contain both abducible predicates and negation-as-failure formulas as in the approaches of [Kakas and Mancarella, 1990; Inoue, 1991a]. Notice that the translation of abducibles by (8.2) is not dual to the translation of negation as failure by (7.6) in Chapter 7. For a negation-as-failure formula $not\, P$, the current model candidate is split into two, one containing $\neg \mathsf{K} P$ and the other containing $\mathsf{K} P$. Here, for an abducible atom H, we generate two model candidates, one containing H itself and the other containing $\neg \mathsf{K} H$. This asymmetry comes from the fact that each hypothesis can be added to model candidates without imposing the condition that it should be derived so that we do not have to check the fixpoint $\mathbf{T}$-condition for the hypothesis. Therefore, when a negation-as-failure formula $not\, H$ mentions an abducible H, it should be split into $\neg \mathsf{K} H$ and H.

While we restrict the abductive framework (Σ, Γ) to a pair of a set of groundable Horn clauses and a set of atomic abducibles, efforts should be devoted to investigating the extensions to non range-restricted cases and to non-Horn clauses and literal abducibles, which can be dealt with by the top-down approach. An example of abduction for non-Horn clauses in a bottom-up manner can be found in Stickel's method

[1991] which uses the contrapositives in the form of definite clauses as in the case of PTTP [Stickel, 1988]. This method may be incorporated for the MGTP+MGTP method. However, since the method does not involve case-splitting, it does not seem to be promising from the viewpoint of parallelism on the MGTP.

8.3.2 Top-down versus Bottom-up

The two reasoning architectures, top-down and bottom-up, are complementary, yet both have merits and demerits for computing abduction. In Section 4.2.1, we pointed out that SOL-resolution is *direct* in the sense that it is both sensitive to the given goal clause and restricted to searching only new characteristic clauses. However, top-down reasoning may result in redundant proofs of subgoals. On the other hand, bottom-up reasoning eliminates redundancy. However, when a query is given, it may prove subgoals unrelated to the proof of the given query.

These facts suggest that it is promising to simulate top-down reasoning by a bottom-up reasoner, or to utilize cached results in top-down reasoning. The former simulation has been proposed for definite Horn databases (without negation as failure) as the Magic Set [Bancilhon *et al.*, 1986] or *upside-down meta-interpretation* [Bry, 1990] methods. As Stickel [1991] argues, this approach is better than the simulation of bottom-up reasoning by a top-down reasoner. This is because caching is more complicated and less effective for abduction since the search space for abduction is larger than for deduction. Therefore, [Stickel, 1991] attempts the upside-down meta-interpretation approach for abduction for Horn and non-Horn clauses. While Stickel does not consider the consistency of abductive explanations in his procedure, his approach is extended for abduction for Horn clauses by incorporating consistency checks for the MGTP+ATMS method [Ohta and Inoue, 1992] and for the MGTP+MGTP method [Inoue *et al.*, 1992b].

An open problem for upside-down meta-interpretation is how to simulate top-down reasoning by the MGTP for non-Horn cases. As stated before, Stickel's method for abduction for non-Horn clauses is not so suitable for parallel execution of the MGTP, further investigation on strategies of combining two reasoning methods is needed for the MGTP. One possible direction is to simulate *near-Horn Prolog* [Reed *et al.*, 1991] on the MGTP. Then, such a "near-Horn Magic Set" method may be easily applied to abduction for non-Horn clauses.

Another important problem which has not been covered in this dissertation but should be addressed in the near future is a development of a neat query answering procedure for nonmonotonic modal logics containing logic programs and deductive databases with negation as failure. While we cannot expect purely top-down proof procedures for these logics, it is an open question how much we can incorporate

the goal information on bottom-up procedures so that the search of the proof is as localized as possible. We should also characterize the largest classes of abductive and nonmonotonic logics that have proof procedures, both syntactically and semantically.

8.4 Other Direction

In the previous section, we have mainly focused on possible extensions of the proposals we gave in the previous chapters from the proof-theoretic viewpoint. However, we have said little about theoretical consideration on the *computational complexity* of hypothetical reasoning except for touching on the *approximation* of consequence-finding in Section 4.3. Since both consequence-finding and model generation are much more complex tasks than proof-finding, most reasoning problems we have considered are computationally intractable, that is, they are NP-hard for propositional cases and are undecidable for general cases. Therefore, there may be many challenging problems concerning the identification of tractable cases of abduction, default reasoning, logic programs and so on (see [Bylander *et al.*, 1991] for abduction). However, we should note that showing that a logical framework of abduction or default reasoning is intractable or undecidable does not mean that it is useless. Since they are intrinsically difficult problems (consider, for instance, scientific discovery as the process of abduction), what we would like to know is that representing a problem in such a framework does not increase the computational complexity of the original problem.

While we have mainly been concerned with *incomplete* knowledge bases in this dissertation, we should also investigate more about how to deal with *inconsistent* knowledge bases. Actually, large-scale knowledge bases or commonsense databases may contain inconsistent information along with incomplete specifications. Notice that, while this kind of inconsistency differs from the problem of multiple, mutually inconsistent augmentations of beliefs (i.e., default extensions) for incomplete knowledge bases, sometimes it can be formalized by a theory formation framework. For example, a method for resolving inconsistency has been described in Section 6.3.2 for extended logic programs. However, we may again encounter alternative ways of removing inconsistencies in the knowledge base, as in the case of multiple extension problems. Thus, this problem may be related to *belief revision* or *update* in databases with integrity constraints. These topics have recently been recognized to be very important for future knowledge based systems.

As argued in Section 2.1, *hypothesis selection*, that is, the problem of how to select good explanations, is also one of the most important problems yet to be solved. This problem is, by its nature, inevitably extra-logical. Therefore, *heuristics* play a central role in hypothesis selection [Inoue, 1988c]. However, we know little about domain-independent heuristics except Occam's razor, and many heuristics must be

domain-dependent. Those studies of heuristics are the central concern of research on *knowledge representation* in AI. Furthermore, once we get some heuristics, we should consider how to restrict the search space of hypotheses using those heuristics. While there are some proposals for this problem of *controlling search* such as [Inoue, 1988b; Inoue and Ohta, 1990], more investigation is needed.

Another key problem is that of automatic generation of a production field $\mathcal{P}$ or hypothesis set Γ. Unfortunately, we have not found any sophisticated way to identify this kind of meta information in general. In abduction, this problem is one of *hypothesis formation*. In fact, one of the big problem of Peirce's [1932] abduction is "how do we come up with those hypotheses Γ?" Peirce argues that this is due to human talent (insight). However, ongoing research issues on induction and analogy in the *machine learning* community would be of help for this problem. It might be that AI itself is trying to achieve a process of hypothesis formation.

Finally, we shall revisit Peirce's theory of scientific reasoning. His theory of scientific discovery relies on the cycle of "experiment, observation, hypothesis generation, hypothesis verification, and hypothesis revision". Peirce mentions that this process involves all modes of reasoning; abduction takes place at the first stage of scientific reasoning, deduction follows to derive the consequences of the hypotheses that were given by abduction, and, finally, induction is used to verify that those hypotheses are true. According to this viewpoint, let us review the logic of abduction:

$$(1) \quad \mathit{Facts} \cup \mathit{Explanation} \models \mathit{Observations}.$$
$$(2) \quad \mathit{Facts} \cup \mathit{Explanation} \quad \mathit{is\ consistent}.$$

A possible interpretation of this form of hypothetical reasoning is now as follows. The formula (1) is the process of abduction, or the fallacy of affirming the consequent. The consistency check (2), on the other hand, is the place where deduction plays a role. Since our knowledge about the world may be incomplete, we should experiment with the consequences using inductive manners in order to verify that the hypotheses are consistent with the knowledge base. At the same time, the process of inductive generalization or the synthesis from examples involves abduction too. When we are given some examples, we first make hypotheses. While previous AI approaches for inductive generalization often enumerated all the possible forms of formulas, abduction would help to restrict the search space. Additional heuristics, once they are formalized, would also be helpful for constructing the hypotheses.

Bibliography

[Arima, 1992] Jun Arima. Logical structure of analogy. In: *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 505–513, Tokyo, Japan, 1992.

[Baker and Ginsberg, 1989] Andrew B. Baker and Matthew L. Ginsberg. A theorem prover for prioritized circumscription. In: *Proceedings of IJCAI-89*, pages 463–467, Detroit, MI, 1989.

[Bancilhon *et al.*, 1986] F. Bancilhon, D. Maier, Y. Sagiv and J.D. Ullman. Magic sets and other strange ways to implement logic programs. In: *Proceedings of the Fifth ACM SIGMOD-SIGACT Symposium on Principles of Database Systems*, pages 1–15, 1986.

[Baral *et al.*, 1991] Chitta Baral, Sarit Kraus and Jack Minker. Combining Multiple Knowledge Bases. *IEEE Trans. Knowledge and Data Engineering*, 3:208–220, 1991.

[Bibel, 1982] Wolfgang Bibel. A comparative study of several proof procedures. *Artificial Intelligence*, 18:269–293, 1982.

[Bidoit, 1989] Nicole Bidoit. Base de données déductives: négation et logique des défaults. Thèse de Docteur d'État, Université de Paris-Sud, Orsay, France, 1989.

[Bossu and Siegel, 1985] Genevieve Bossu and Pierre Siegel. Saturation, nonmonotonic reasoning, and the closed-world assumption. *Artificial Intelligence*, 25:13–63, 1985.

[Brewka, 1989a] Gerhard Brewka. Nonmonotonic reasoning: from theoretical foundations towards efficient computation. Ph.D. Dissertation, University of Hamburg, Hamburg, Germany, 1989.

[Brewka, 1989b] Gerhard Brewka. Preferred subtheories: an extended logical framework for default reasoning. In: *Proceedings of IJCAI-89*, pages 1043–1048, Detroit, MI, 1989.

[Bry, 1990] François Bry. Query evaluation in recursive databases: bottom-up and top-down reconciled. *Data & Knowledge Engineering*, 5:289–312, 1990.

[Bylander *et al.*, 1991] Tom Bylander, Dean Allemang, Michael C. Tanner and John R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49:25–60, 1991.

[Chang and Lee, 1973] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[Charniak and McDermott, 1985] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA, 1985.

[Clark, 1978] Keith L. Clark. Negation as failure. In: Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 119–140, Plenum Press, New York, 1978.

[Console *et al.*, 1991] Luca Console, Daniele Theseider Dupre and Pietro Torasso. On the relationship between abduction and deduction. *J. Logic Computation*, 1:661–690, 1991.

[Cox and Pietrzykowski, 1986] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and applications. In: *Proceedings of the Eighth International Conference on Automated Deduction*, Lecture Notes in Computer Science, 230, pages 608–621, Springer-Verlag, 1986.

[de Kleer, 1986] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[de Kleer, 1988] Johan de Kleer. A general labeling algorithm for assumption-based truth maintenance. In: *Proceedings of AAAI-88*, pages 188–192, St. Paul, MN, 1988.

[de Kleer, 1989] Johan de Kleer. A comparison of ATMS and CSP techniques. In: *Proceedings of IJCAI-89*, pages 290–296, Detroit, MI, 1989.

[de Kleer *et al.*, 1990] Johan de Kleer, Alan K. Mackworth and Raymond Reiter. Characterizing diagnoses. In: *Proceedings of AAAI-90*, pages 324–329, Boston, MA, 1990.

[Demolombe and Fariñas del Cerro, 1991] Robert Demolombe and Luis Fariñas del Cerro. An inference rule for hypothesis generation. In: *Proceedings of IJCAI-91*, pages 152–157, Sydney, Australia, 1991.

[Doyle, 1979] Jon Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[Dressler, 1989] Oskar Dressler. An extended basic ATMS. In: *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, Lecture Notes in Artificial Intelligence, 346, pages 143–163, Springer-Verlag, 1989.

[Elkan, 1990] Charles Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence*, 43:219–234, 1990.

[Enderton, 1972] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.

[Eshghi, 1990] Kave Eshghi. Computing stable models by using the ATMS. In: *Proceedings of AAAI-90*, pages 272–277, Boston, MA, 1990.

[Eshghi and Kowalski, 1989] K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In: Giorgio Levi and Maurizio Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming* (Lisbon, Portugal), pages 234–254, MIT Press, Cambridge, MA, 1989.

[Etherington, 1988] David W. Etherington. *Reasoning with Incomplete Information.* Pitman, London, 1988.

[Finger, 1987] Joseph J. Finger. Exploiting constraints in design synthesis. Ph.D. Dissertation, Technical Report STAN-CS-88-1204, Department of Computer Science, Stanford University, Stanford, CA, April 1987.

[Fernández and Minker, 1991] José Alberto Fernández and Jack Minker. Bottom-up evaluation of hierarchical disjunctive deductive databases. In: Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pages 660–675, MIT Press, Cambridge, MA, 1991.

[Fernández and Minker, 1992] José Alberto Fernández and Jack Minker. Disjunctive Deductive Databases. In: *Proceedings of International Conference on Logic Programming and Automated Reasoning* (St. Petersbyrg, Russia), Lecture Notes in Artificial Intelligence, 624, pages 332–356, Springer-Verlag, 1992.

[Fujita and Hasegawa, 1991] Hiroshi Fujita and Ryuzo Hasegawa. A model generation theorem prover in KL1 using a ramified-stack algorithm. In: Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pages 535–548, MIT Press, Cambridge, MA, 1991.

[Geffner, 1990] Hector Geffner. Causal theories for nonmonotonic reasoning. In: *Proceedings of AAAI-90*, pages 524–530, Boston, MA, 1990.

[Gelfond, 1990] Michael Gelfond. Epistemic approach to formalization of commonsense reasoning. Technical Report, Computer Science Department, University of Texas at El Paso, El Paso, TX, 1990.

[Gelfond, 1991] Michael Gelfond. Strong introspection. In: *Proceedings of AAAI-91*, pages 386–391, Anaheim, CA, 1991.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In: Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming* (Seattle, WA), pages 1070–1080, MIT Press, Cambridge, MA, 1988.

[Gelfond and Lifschitz, 1990] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In: David H.D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming* (Jerusalem, Israel), pages 579–597, MIT Press, Cambridge, MA, 1990.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Gelfond *et al.*, 1991] Michael Gelfond, Vladimir Lifschitz, Halina Przymusińska and Mirosław Truszczyński. Disjunctive defaults. In: *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 230–237, Cambridge, MA, 1991.

[Gelfond *et al.*, 1989] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38:75–94, 1989.

[Gelfond *et al.*, 1990] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. On the relationship between CWA, minimal model, and minimal Herbrand model semantics. *Int. J. Intelligent Systems*, 5:549–564, 1990.

[Genesereth and Nilsson, 1987] Michael Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1987.

[Ginsberg, 1988] Matthew L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[Ginsberg, 1989] Matthew L. Ginsberg. A circumscriptive theorem prover. *Artificial Intelligence*, 39:209–230, 1989.

[Goebel *et al.*, 1986] Randy Goebel, Koichi Furukawa and David Poole. Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. In: Ehud Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming* (London, UK), pages 211–222, Lecture Notes in Computer Science, 225, Springer-Verlag, 1986.

[Giordano and Martelli, 1990] Laura Giordano and Alberto Martelli. Generalized stable models, truth maintenance and conflict resolution. In: David H.D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference*

on Logic Programming (Jerusalem, Israel), pages 427–441, MIT Press, Cambridge, MA, 1990.

[Green, 1969a] Cordell Green. Theorem-proving by resolution as a basis for question-answering systems. In: B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 183–205, Edinburgh University Press, Edinburgh, 1969.

[Green, 1969b] Cordell Green. Application of theorem proving to problem solving. In: *Proceedings of IJCAI-69*, pages 219–239, Washington, D.C., 1969.

[Guerreiro *et al.*, 1990] Ramiro A. de T. Guerreiro, Marco A. Casanova and Andrea S. Hemerly. Contributions to a proof theory for generic defaults. In: *Proceedings of ECAI-90*, pages 213–218, Stockholm, Sweden, 1990.

[Hanks and McDermott, 1986] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics and the frame problem. In: *Proceedings of AAAI-86*, pages 328–333, Philadelphia, PA, 1986.

[Helft *et al.*, 1989] Nicolas Helft, Katsumi Inoue and David Poole. Extracting answers in circumscription. ICOT Technical Memorandum TM-855, ICOT, Tokyo, Japan, December 1989.

[Helft *et al.*, 1991] Nicolas Helft, Katsumi Inoue and David Poole. Query answering in circumscription. In: *Proceedings of IJCAI-91*, pages 426–431, Sydney, Australia, 1991.

[Helft and Konolige, 1990] Nicolas Helft and Kurt Konolige. Plan recognition as abduction and relevance. Technical Report, SRI International, Menlo Park, CA, November 1990.

[Hempel, 1966] Carl Gustav Hempel. *Philosophy of Natural Science*. Prentice-Hall, New Jersey, 1966.

[Hobbs *et al.*, 1988] Jerry R. Hobbs, Mark Stickel, Paul Martin and Douglas Edwards. Interpretation as abduction. In: *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 95–103, Buffalo, NY, 1988.

[Inoue, 1988a] Katsumi Inoue. On the semantics of hypothetical reasoning and truth maintenance. ICOT Technical Report TR-357, ICOT, Tokyo, Japan, March 1988.

[Inoue, 1988b] Katsumi Inoue. Pruning search trees in assumption-based reasoning. In: *Proceedings of the Eighth International Workshop on Expert Systems & their Applications*, pages 133–151, Avignon, France, 1988.

[Inoue, 1988c] Katsumi Inoue. Problem solving with hypothetical reasoning. In: *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, pages 1275–1281, Tokyo, Japan, 1988.

[Inoue, 1989] Katsumi Inoue. Generalizing the ATMS: a model-based approach (preliminary report). *IPSJ SIG Reports*, SIG AI 63–3, pages 21–28, Information Processing Society of Japan, March 1989. Also, in: ICOT Technical Memorandum TM-621, ICOT, February 1989.

[Inoue, 1990a] Katsumi Inoue. Procedural interpretation for an extended ATMS. ICOT Technical Report TR-547, ICOT, Tokyo, Japan, March 1990.

[Inoue, 1990b] Katsumi Inoue. An abductive procedure for the CMS/ATMS. In: João P. Martins and Michael Reinfrank, editors, *Truth Maintenance Systems, Proceedings of the ECAI-90 Workshop* (Stockholm, Sweden, August 1990), Lecture Notes in Artificial Intelligence, 515, pages 34–53, Springer-Verlag, 1991.

[Inoue, 1991a] Katsumi Inoue. Extended logic programs with default assumptions. In: Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pages 490–504, MIT Press, Cambridge, MA, 1991. Also, an extended version: Hypothetical reasoning in logic programs. ICOT Technical Report TR-691, ICOT, Tokyo, Japan, August 1991.

[Inoue, 1991b] Katsumi Inoue. Consequence-finding based on ordered linear resolution. In: *Proceedings of IJCAI-91*, pages 158–164, Sydney, Australia, 1991.

[Inoue, 1992a] Katsumi Inoue. Principles of abduction. *J. Japanese Society for Artificial Intelligence*, 7(1):48–59, January 1992 (in Japanese).

[Inoue, 1992b] Katsumi Inoue. Linear resolution for consequence finding. *Artificial Intelligence*, 56:301–353, 1992.

[Inoue and Helft, 1990] Katsumi Inoue and Nicolas Helft. On theorem provers for circumscription. In: Peter F. Patel-Schneider, editor, *Proceedings of the Eighth Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (Ottawa, Ontario, Canada), pages 212–219, Morgan Kaufmann, Palo Alto, CA, 1990. Also, a revised version is to appear in: Makoto Amamiya, editor, *Logic Programming, Proceedings of the Ninth Conference* (Tokyo, Japan, July 1990), Lecture Notes in Artificial Intelligence, Springer-Verlag.

[Inoue *et al.*, 1992a] Katsumi Inoue, Miyuki Koshimura and Ryuzo Hasegawa. Embedding negation as failure into a model generation theorem prover. In: Deepak Kapur, editor, *Proceedings of the Eleventh International Conference on Automated Deduction* (Saratoga Springs, NY), Lecture Notes in Artificial Intelligence, 607, pages 400–415, Springer-Verlag, 1992.

[Inoue and Ohta, 1990] Katsumi Inoue and Yoshihiko Ohta. Making dependency-directed search hierarchical. In: *Proceedings of the Pacific Rim International Conference on Artificial Intelligence '90*, pages 450–455, Nagoya, Japan, 1990.

[Inoue *et al.*, 1992b] Katsumi Inoue, Yoshihiko Ohta, Ryuzo Hasegawa and Makoto Nakashima. Hypothetical reasoning systems on the MGTP. ICOT Technical Report TR-763, ICOT, Tokyo, Japan, August 1992 (in Japanese).

[Inoue and Sakama, 1992] Katsumi Inoue and Chiaki Sakama. A uniform approach to fixpoint characterization of disjunctive and general logic programs. ICOT Technical Report, ICOT, Tokyo, Japan, October 1992.

[Iwanuma *et al.*, 1989] Kouji Iwanuma, Masateru Harao and Shoichi Noguchi. A computation method for parallel circumscription based on equivalent transformation of queries. *Reports of the Group for Computation IEICE Japan*, COMP89-42, pages 21–30, 1989 (in Japanese).

[Junker, 1989] Ulrich Junker. A correct non-monotonic ATMS. In: *Proceedings of IJCAI-89*, pages 1049–1054, Detroit, MI, 1989.

[Junker and Konolige, 1990] Ulrich Junker and Kurt Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In: *Proceedings of AAAI-90*, pages 278–283, Boston, MA, 1990.

[Kakas and Mancarella, 1990] A.C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In: *Proceedings of ECAI-90*, pages 385–391, Stockholm, Sweden, 1990.

[Kakas *et al.*, 1992] A.C. Kakas, R.A. Kowalski and F. Toni. Abductive logic programming. A survey paper, Imperial College of Science, Technology and Medicine, London, UK, 1992.

[Kean and Tsiknis, 1990] Alex Kean and George Tsiknis. An incremental method for generating prime implicants/implicates. *J. Symbolic Computation*, 9:185–206, 1990.

[Konolige, 1988] Kurt Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.

[Koshimura and Hasegawa, 1991] Miyuki Koshimura and Ryuzo Hasegawa. Modal propositional tableaux in a model generation theorem prover. *Proceedings of the Tenth Logic Programming Conference*, pages 43–51, Tokyo, Japan, July 1991 (in Japanese).

[Kowalski, 1989] Robert Kowalski. The treatment of negation in logic programs for representing legislation. In: *Proceedings of the Second International Conference on Artificial Intelligence and Law*, pages 11–15, Vancouver, BC, 1989.

[Kowalski, 1990] Robert A. Kowalski. Problems and promises of computational logic. In: J.W. Lloyd, editor, *Proceedings of the Symposium on Computational Logic* (Brussels, Belguim), pages 1–36, Springer-Verlag, 1990.

[Kowalski and Hayes, 1969] R. Kowalski and P.J. Hayes. Semantic trees in automatic theorem-proving. In: B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 87–101, Edinburgh University Press, Edinburgh, 1969.

[Kowalski and Kuhner, 1971] Robert Kowalski and Donald G. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.

[Kowalski and Sadri, 1990] Robert A. Kowalski and Fariba Sadri. Logic programs with exceptions. In: David H.D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming* (Jerusalem, Israel), pages 598–613, MIT Press, Cambridge, MA, 1990.

[Lee, 1967] Richard Char-Tung Lee. A completeness theorem and computer program for finding theorems derivable from given axioms. Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA, December 1967.

[Levesque, 1989] Hector J. Levesque. A knowledge-level account of abduction (preliminary version). In: *Proceedings of IJCAI-89*, pages 1061–1067, Detroit, MI, 1989.

[Levesque, 1990] Hector J. Levesque. All I know: a study of autoepistemic logic. *Artificial Intelligence*, 42:263–309, 1990.

[Lifschitz, 1985] Vladimir Lifschitz. Computing circumscription. In: *Proceedings of IJCAI-85*, pages 121–127, Los Angeles, CA, 1985.

[Lifschitz, 1986] Vladimir Lifschitz. On the satisfiability of circumscription. *Artificial Intelligence*, 28:17–27, 1986.

[Lifschitz, 1991] Vladimir Lifschitz. Nonmonotonic databases and epistemic queries. In: *Proceedings of IJCAI-91*, pages 381–386, Sydney, Australia, 1991.

[Lin, 1991] Fangzhen Lin. A study of nonmonotonic reasoning. Ph.D. Dissertation, Technical Report STAN-CS-91-1385, Department of Computer Science, Stanford University, Stanford, CA, August 1991.

[Lin and Goebel, 1989] Dekang Lin and Randy Goebel. Computing circumscription of ground theories with Theorist. Technical Report TR-89-26, Department of Computer Science, The University of Alberta, Edmonton, Alberta, October 1989.

[Lloyd, 1984] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, First Edition, 1984; Second Edition, 1987.

[Lloyd and Topor, 1984] J.W. Lloyd and R.W. Topor. Making Prolog more expressive. *J. Logic Programming*, 3:225–240, 1984.

[Loveland, 1969] Donald W. Loveland. A simplified format for the model elimination theorem-proving procedure. *J. ACM*, 16:349–363, 1969.

[Loveland, 1978] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis*. North-Holland, Amsterdam, 1978.

[Manthey and Bry, 1988] Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In: *Proceedings of the Ninth International Conference on Automated Deduction*, Lecture Notes in Computer Science, 310, pages 415–434, Springer-Verlag, 1988.

[Marek and Truszczyński, 1989] Wiktor Marek and Mirosław Truszczyński. Relating autoepistemic and default logic. In: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 276–288, Toronto, Ontario, Canada, 1989.

[Martins, 1991] João P. Martins. The truth, the whole truth, and nothing but the truth: an indexed bibliography to the literature of truth maintenance systems. *AI Magazine*, Special Issue:7–25, 1991.

[McCarthy, 1977] John McCarthy. Epistemological problems of artificial intelligence. In: *Proceedings of IJCAI-77*, pages 1038–1044, Cambridge, MA, 1977.

[McCarthy, 1980] John McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[McCarthy and Hayes, 1969] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In: B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502, Edinburgh University Press, Edinburgh, 1969.

[McDermott and Doyle, 1980] Drew McDermott and Jon Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[Minicozzi and Reiter, 1972] Eliana Minicozzi and Raymond Reiter. A note on linear resolution strategies in consequence-finding. *Artificial Intelligence*, 3:175–180, 1972.

[Minker, 1982] Jack Minker. On indefinite databases and the closed world assumption. In: *Proceedings of the Sixth International Conference on Automated Deduction*, Lecture Notes in Computer Science, 138, pages 292–308, Springer-Verlag, 1982.

[Minker and Rajasekar, 1990] Jack Minker and Arcot Rajasekar. A fixpoint semantics for disjunctive logic programs. *J. Logic Programming*, 9:45–74, 1990.

[Moore, 1985] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[Morris, 1987] Paul H. Morris. Curing anomalous extensions. In: *Proceedings of AAAI-87*, pages 437–442, Seattle, WA, 1987.

[Morris, 1989] Paul H. Morris. Autoepistemic stable closures and contradiction resolution. In: *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, Lecture Notes in Artificial Intelligence, 346, pages 60–73, Springer-Verlag, 1989.

[Nitta *et al.*, 1992] K. Nitta, Y. Ohtake, S. Maeda, M. Ono, H. Ohsaki and K. Sakane. HELIC-II: a legal reasoning system on the parallel inference machine. In: *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 1115–1124, Tokyo, Japan, 1992.

[Ohta and Inoue, 1990] Yoshihiko Ohta and Katsumi Inoue. A forward-chaining multiple context reasoner and its application to logic design. In: *Proceedings of the Second IEEE International Conference on Tools for Artificial Intelligence*, pages 386–392, Herndon, VA, 1990.

[Ohta and Inoue, 1991a] Yoshihiko Ohta and Katsumi Inoue. An efficient inference method for forward-chaining hypothetical reasoning with the ATMS. *J. Japanese Society for Artificial Intelligence*, 6(2):247–259, March 1991 (in Japanese).

[Ohta and Inoue, 1991b] Yoshihiko Ohta and Katsumi Inoue. An incremental hypothesis-based forward-reasoning system. *J. Japanese Society for Artificial Intelligence*, 6(4):532–544, July 1991 (in Japanese).

[Ohta and Inoue, 1992] Yoshihiko Ohta and Katsumi Inoue. A forward-chaining hypothetical reasoner based on upside-down meta-interpretation. In: *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 522–529, Tokyo, Japan, 1992.

[O'Rorke, 1990] Paul O'Rorke, editor. *Working Notes of the AAAI Spring Symposium on Automated Abduction*, Stanford, CA, March 1990.

[Peirce, 1932] Charles Sanders Peirce. *Elements of Logic*. In: Charles Hartshorne and Paul Weiss, editors, *Collected Papers of Charles Sanders Peirce*, Volume II, Harvard University Press, Cambridge, MA, 1932.

[Pereira *et al.*, 1991a] Luís Moniz Pereira, Joaquim Nunes Aparício and José Júlio Alferes. Nonmonotonic reasoning with well founded semantics. In: Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pages 475–489, MIT Press, Cambridge, MA, 1991.

[Pereira *et al.*, 1991b] Luís Moniz Pereira, Joaquim Nunes Aparício and José Júlio Alferes. Contradiction removal within well-founded semantics. In: Anil Nerode, Wiktor Marek and V.S. Subrahmanian, editors, *Proceedings of the First International Workshop on Logic Programming and Non-monotonic Reasoning* (Washington, D.C.), pages 105–119, MIT Press, Cambridge, MA, 1991.

[Poole, 1988] David Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[Poole, 1989] David Poole. Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence*, 5:97–110, 1989.

[Poole, 1991] David Poole. Compiling a default reasoning system into Prolog. *New Generation Computing*, 9:3–38, 1991.

[Poole *et al.*, 1987] David Poole, Randy Goebel and Romas Aleliunas. Theorist: a logical reasoning system for defaults and diagnosis. In: Nick Cercone and Gordon McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352, Springer-Verlag, New York, 1987.

[Pople, 1973] Harry E. Pople, Jr. On the mechanization of abductive logic. In: *Proceedings of IJCAI-73*, pages 147–152, Stanford, CA, 1973.

[Provan, 1990] Gregory M. Provan. The computational complexity of multiple-context truth maintenance systems. In: *Proceedings of ECAI-90*, pages 522–527, Stockholm, Sweden, 1990.

[Przymusinski, 1989] Teodor C. Przymusinski. An algorithm to compute circumscription. *Artificial Intelligence*, 38:49–73, 1989.

[Przymusinski, 1990a] Teodor C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In: David H.D. Warren and Peter Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming* (Jerusalem, Israel), pages 459–477, MIT Press, Cambridge, MA, 1990.

[Przymusinski, 1990b] Teodor C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In: Saumya Debray and Manuel Hermenegildo, editors, *Proceedings of the 1990 North American Conference on Logic Programming* (Austin, TX), pages 40–59, MIT Press, Cambridge, MA, 1990.

[Quine, 1952] W.V.O. Quine. The problem of simplifying truth functions. *Am. Math. Monthly*, 59:521–531, 1952.

[Reed *et al.*, 1991] David W. Reed, Donald W. Loveland and Bruce T. Smith. An alternative characterization of disjunctive logic programs. In: Vijay Saraswat and Kazunori Ueda, editors, *Proceedings of the 1991 International Logic Programming Symposium* (San Diego, CA), pages 54–68, MIT Press, Cambridge, MA, 1991.

[Reiter, 1971] Raymond Reiter. Two results on ordering for resolution with merging and linear format. *J. ACM*, 18:630–646, 1971.

[Reiter, 1978] Raymond Reiter. On closed world data bases. In: Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 55–76, Plenum Press, New York, 1978.

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Reiter, 1984] Raymond Reiter. Towards a logical reconstruction of relational database theory. In: M.L. Brodie, J. Mylopoulous and J.W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pages 191–238, Springer-Verlag, New York, 1984.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[Reiter, 1990] Raymond Reiter. What should a database know? Technical Reports on Knowledge Representation and Reasoning KRR-TR-90-5, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1990.

[Reiter and de Kleer, 1987] Raymond Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: preliminary report. In: *Proceedings of AAAI-87*, pages 183–187, Seattle, WA, 1987.

[Rescher, 1964] Nicholas Rescher. *Hypothetical Reasoning.* North-Holland, Amsterdam, 1964.

[Robinson, 1965] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12:23–41, 1965.

[Saccà and Zaniolo, 1990] Domenico Saccà and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In: *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205–229, Nashville, TN, 1990.

[Sadri and Kowalski, 1987] Fariba Sadri and Robert Kowalski. A theorem proving approach to database integrity. In: Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 313–362, Morgan Kaufmann, Los Altos, CA, 1987.

[Sartor, 1991] Giovanni Sartor. The structure of norm conditions and nonmonotonic reasoning in law. In: *Proceedings of the Third International Conference on Artificial Intelligence and Law*, pages 155–164, Oxford, England, 1991.

[Satoh and Iwayama, 1991] Ken Satoh and Noboru Iwayama. Computing abduction by using the TMS. In: Koichi Furukawa, editor, *Proceedings of the Eighth International Conference on Logic Programming* (Paris, France), pages 505–518, MIT Press, Cambridge, MA, 1991.

[Sattar *et al.*, 1990] Abdul Sattar, Scott Goodwin and Randy Goebel. What's in a consistency tree: a uniform treatment of theory selection. In: *Proceedings of the Pacific Rim International Conference on Artificial Intelligence '90*, pages 426–431, Nagoya, Japan, 1990.

[Schlipf, 1986] John S. Schlipf. How uncomputable is general circumscription? In: *Proceedings of the First IEEE Symposium on Logic in Computer Science*, pages 92–95, Cambridge, MA, 1986.

[Selman and Levesque, 1990] Bart Selman and Hector J. Levesque. Abductive and default reasoning: a computational core. In: *Proceedings of AAAI-90*, pages 343–348, Boston, MA, 1990.

[Shoham, 1987] Yoav Shoham. A semantical approach to nonmonotonic logics. In: *Proceedings of IJCAI-87*, pages 388–392, Milan, Italy, 1987.

[Shostak, 1976] Robert E. Shostak. Refutation graphs. *Artificial Intelligence*, 7:51–64, 1976.

[Siegel, 1987] Pierre Siegel. Représentation et utilisation de la connaissance en calcul propositionnel. Thèse d'État, Université d'Aix-Marseille II, Luminy, France, 1987 (in French).

[Siegel and Schwind, 1991] Pierre Siegel and Camilla Schwind. Hypothesis theory for nonmonotonic reasoning. In: *Proceedings of the Workshop on Nonstandard Queries and Nonstandard Answers*, pages 189–210, Toulouse, France, 1991.

[Slagle *et al.*, 1969] J.R. Slagle, C.L. Chang and R.C.T. Lee. Completeness theorems for semantic resolution in consequence-finding. In: *Proceedings of IJCAI-69*, pages 281–285, Washington, D.C., 1969.

[Stickel, 1988] Mark E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *J. Automated Reasoning*, 4:353–380, 1988.

[Stickel, 1989] Mark E. Stickel. Rationale and methods for abductive reasoning in natural-language interpretation. In: R. Studer, editor, *Natural Language and Logic, Proceedings of the International Scientific Symposium* (Hamburg, Germany, May 1989), Lecture Notes in Artificial Intelligence, 459, pages 233–252, Springer-Verlag, 1990.

[Stickel, 1991] Mark E. Stickel. Upside-down meta-interpretation of the model elimination theorem-proving procedure for deduction and abduction. ICOT Technical Report TR-664, ICOT, Tokyo, Japan, 1991.

[Tison, 1967] Pierre Tison. Generalized consensus theory and application to the minimization of boolean functions. *IEEE Transactions on Electronic Computers*, 16:446–456, 1967.

[Ueda and Chikayama, 1990] K. Ueda and T. Chikayama, Design of the kernel language for the parallel inference machine, *Computer Journal*, 33:494–500, 1990.

[van Emden and Kowalski, 1976] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23:733–742, 1976.

[van Gelder *et al.*, 1991] Allen van Gelder, Kenneth A. Ross and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38:620–650, 1991.

[Wakayama and Payne, 1988] T. Wakayama and T.H. Payne. Case inference in resolution-based languages. In: *Proceedings of the Ninth International Conference on Automated Deduction*, Lecture Notes in Computer Science, 310, pages 313–322, Springer-Verlag, 1988.